MS-DOS® Series

PC-9800 Series

For Personal Computers Running the MS-DOS®

Version 1.0

Microsoft® Visual Basic™

Programming System for MS-DOS®

マイクロソフト ビジュアル ベーシック　プログラミング システム

機種依存情報

Microsoft® VISUAL BASIC™

Microsoft® Visual Basic™

Programming System for MS-DOS®

マイクロソフト ビジュアル ベーシック　プログラミング システム

機種依存情報

Microsoft® VISUAL BASIC™

ご注意

① このソフトウェアの著作権は、米国マイクロソフト社にあります。
② このソフトウェアおよびマニュアルの一部または全部を無断で使用、複製することはできません。
③ ソフトウェアは、コンピュータ 1 台につき 1 セット購入が原則になっております。
④ このソフトウェアおよびマニュアルは、本製品の使用許諾契約書のもとでのみ使用することができます。
⑤ このソフトウェアおよびマニュアルを運用した結果の影響については、いっさい責任を負いかねますのでご了承ください。

MS, Microsoft, MS-DOS は米国マイクロソフト社の登録商標です。
QuickBasic, Visual Basic, Windows は米国マイクロソフト社の商標です。
386 は米国インテル社の商標です。
OS/2 は米国マイクロソフト社がライセンスされたオペレーティングシステムです。

Microsoft Visual Basic for MS-DOS のマニュアルは、 Microsoft Word for Windows Version 1.2A を使用して製作しました。

目次

はじめに v

1 PC-9801 シリーズ固有の仕様 1

1.1 画面表示とスクリーン モード 1

SCREEN ステートメントによるスクリーン モードの設定 2

色値，パレット番号，色番号 3

色の指定方法 (COLOR ステートメント) 6

グラフィックス スクリーン ページ 7

画面行数とフォーム，INPUTBOX$，MSGBOX の表示 8

テキスト画面のみを使うときの制限 8

1.2 キーボードからの入力 9

キーの名称 9

KeyDown, KeyUp イベントの *KeyCode* 引数 9

KEY ステートメントの *keyboardflag%* と *code%* 引数 11

INKEY$ 関数が返す値 11

ファンクション キー [F16] 13

1.3 その他 13

ASCII コード, 漢字連続コード以外のコードに対応する文字 13

タイマーの精度 15

UTILITY.BAS サンプル アプリケーション 15

2 グラフィックス 17

2.1 スクリーン モードの設定 17

2.2 ピクセルと画面座標 18

2.3 点，線，ボックスなどの基本図形の描画 19

PSET と PRESET による点の描画 19

LINE による線の描画 20

2.4 CIRCLE による円と楕円の描画 24

円の描画 24

楕円の描画 24

円弧の描画 25

扇形の描画 27

2.5 DRAW: グラフィックス マクロ言語 28

2.6 グラフィックス ビューポートの定義 30

2.7 ビューポート座標の再定義 33

座標値の順序 35

ビューポート座標と物理座標の獲得 36

2.8 色の選択 37
パレットとパレット番号 37
パレットの設定の変更 37
2.9 図形の塗りつぶし 39
色による塗りつぶし 39
塗りつぶし模様の指定: タイリング 40
2.10 基本的なアニメーション技法 46
イメージの保存: GET 46
イメージの移動: PUT 48
GET と PUT を使ったアニメーション 51
2.11 アプリケーションの例 56
棒グラフの作成 56
フラクタルのグラフィックス 61
パターン エディタ 65

3 QB4.5 および BC7.1 との互換性 71
3.1 予約語の互換性 71
3.2 各製品の設計上の相違 73
3.3 各製品の主な仕様の相違 74
3.4 ステートメントおよび関数の機能の互換性 75

4 QB4.5 および BC7.1 アプリケーションの移植 79
4.1 一般的な考え方 79
4.2 Quick ライブラリ (.QLB) 80
QB4.5 と BC7.1 で提供されている Quick ライブラリ 80
Basic で書かれた Quick ライブラリ 81
C で書かれた Quick ライブラリ 81
MASM で書かれた Quick ライブラリ 81
4.3 オブジェクト モジュール ライブラリ (.LIB) 82
4.4 アプリケーションのロード 83
4.5 予約語のチェック 85
4.6 ダイアログ ボックスの追加 85
再設計のレベル 86
段階的な改良 86
MSGBOX と INPUTBOX$ を使ったダイアログ ボックス 89
コモン ダイアログ ツールキットを使ったダイアログ ボックス 92
フォームを使ったダイアログ ボックス 92
4.7 インターフェイスの再設計 94

はじめに

必要なシステム構成

アプリケーションの作成に必要なシステム

CPU	80286 以上を搭載する PC-9801 シリーズ (ただし VX0/2/4 とハイレゾリューション モードを除く)
主な対象機種	VX01/21/41, UX21/41, RX, RS, RA, RL, DX, DS, DA, FX, FS, FA, H98, H98s, NS, NS/E, NS/T
OS	MS-DOS バージョン 3.1 以上
必要メモリ	メイン メモリ 640K バイト (空きメモリ 480K バイト以上) XMS または EMS メモリ 1M バイト以上
必要ディスク装置	1 台以上の 1.25M バイトのフロッピー ディスク ドライブ 5M バイト以上のハード ディスク空き容量
その他の必要な周辺装置	アナログ ディスプレイ，マウス
推奨システム	386 以上の CPU を装備する機種 1M バイトの XMS メモリ 1M バイト以上の EMS メモリ
使用可能な日本語入力システム	MS KANJI API に準拠した日本語入力システム

作成した実行可能 (.EXE) ファイルの実行に必要なシステム

CPU	アプリケーションの作成に必要なシステムと同様
OS	アプリケーションの作成に必要なシステムと同様
必要なメモリ	作成したアプリケーションに依存
その他	アナログ ディスプレイ，マウス

ランタイム モジュールの再配布

Visual Basic for MS-DOS で作成したプログラムを、Visal Basic for MS-DOS がインストールされていないシステムで実行するときには、ランタイム モジュール VBDRT10.EXE が必要になる場合があります。

Visual Basic for MS-DOS で作成したプログラムを第三者に配布する場合、ソフトウェアライセンス契約に記載されている条件のもとで、これらのランタイム モジュールを添付して配布することができます。ただしランタイムモジュールの再配布は、同時に配布するアプリケーションの一部として使われる場合に限ります。

UTILITY ツールキット

PC-9801 シリーズ特有のツールキットとして、画面のハード コピーなどの機能を提供する UTILITY.QLB, UTILITY.LIB と、それを構成するファイルである UTILITY.BAS, UTILASM.OBJ が用意されています。使い方について詳しくは、このマニュアルの第 1 章「PC-9801 シリーズ固有の仕様」を参照してください。

Visual Basic for MS-DOS のマニュアル構成

Visual Basic for MS-DOS のパッケージには、次の 3 冊のマニュアルと、オンライン ヘルプが含まれています。

- 『プログラミング ガイド』

  Visual Basic for MS-DOS を使ってどのようにアプリケーションを作成するかを、例をまじえながら説明します。実際に簡単なアプリケーションを作成しながら、プログラミング環境でコードを記述する方法と、フォーム デザイナでフォームを設計，作成する方法を説明した後、Visual Basic for MS-DOS がサポートしている各機能をどのように活用していくかを順に触れて行きます。このマニュアルを読んで、コードの動作と意味について理解してください。

- 『リファレンス』

  Visual Basic for MS-DOS の言語仕様に関するリファレンスです。

- 『機種依存情報』(このマニュアル)

  グラフィックスやキーボード スキャン コードなどの、ハードウェアに依存する機能について説明しています。例えば、グラフィックスを表示するときに必要なスクリーン モードの種類と値などは、このマニュアルに載っています。『プログラミング ガイド』および『リファレンス』には、ハードウェアに依存した機能の詳細は記載されておらず、このマニュアルを参照するようになっています。

  また、Microsoft の以前のバージョンの Basic との互換性についても、このマニュアルを参照してください。

- オンライン ヘルプ

  Visual Basic for MS-DOS の操作方法と言語仕様についてのリファレンスです。マニュアルを読んでVisual Basic の基本的な内容について理解した後は、構文などの細かい仕様やメニュー コマンドの機能を調べるために、このオンライン ヘルプを利用してください。オンライン ヘルプのトピックを表示するには、キーワードにカーソルを合わせて [F1] キーを押します。また、ダイアログ ボックスなどで <ヘルプ> ボタンがあれば、より詳しい説明を含むトピックが用意されています。ほとんどの関数やステートメントの説明には、プログラム例が含まれます。これらの例は、コード ウィンドウに複写/貼り付けして実行でき、アプリケーションで利用することができます。

このマニュアルの構成

このマニュアルは 4 章から構成されています。

- 第 1 章は、PC-9801 シリーズ用の Visual Basic for MS-DOS 固有の仕様に関して説明しています。『プログラミング ガイド』および『リファレンス』は機種に依存しない記述になっており、PC-9801 シリーズに固有の機能は、この章を参照するようになっています。なお、オンライン ヘルプは、PC-9801 シリーズの機能に合わせて記述されています。
- 第 2 章は、グラフィックス機能の説明です。グラフィックスの機能は機種に依存する部分が多いため、ここにまとめてあります。
- 第 3 章と第 4 章では、QuickBasic バージョン 4.5 および Microsoft Basic Professional Development System バージョン 7.1 (BC7.1) との互換性について説明しています。第 3 章で予約語から見た違いに触れた後、第 4 章では QuickBasic または Microsoft Basic で作成したアプリケーションを Visual Basic for MS-DOS に移植する一般的な方法について説明しています。

マニュアルの表記規則

このマニュアルは、次の表記規則に従って記述されています。

表記	説明
SUB, IF, CHDIR, PRINT, TIME$	大文字のボールド (太字) は、Visual Basic for MS-DOS の予約語であることを表します。
BackColor, Click	プロパティとイベントの名前は、大文字で始まります。
setup	プログラマが実際に入力する単語はボールドで示します。
eventname	構文中のイタリック (斜体) は、ファイル名のように適当な文字列に置き換える必要のあるものを表します。単語の最後の文字は、指定するデータ型を表すこともあります。
[*expressionlist*]	構文中の角カッコ ([]) は、囲まれた項目が省略できることを表します。

{**While**|**Until**}	構文中の大カッコ ({}) と縦棒の組み合わせは、複数の項目から選択する必要があることを表します。全体が角カッコ ([]) で囲まれていなければ、いずれか 1 つの項目を選ばなければなりません。

`SUB HelloButton_Click ()`
`  Readout.Text = "Hello, world!"`
`END SUB`
	この書体は、変数，コード，例，エラーメッセージを表します。

`Readout.Text = "This should all be` `⇒ on one line in the Code window."`	行連結文字 (⇒) はコードが次の行に続くことを表します。実際にプログラムとして入力するときは、1 行で入力します。
CONSTANT.BI, F1_KEY	ファイル名や定数名は、すべて大文字で示してあります。
[ENTER]	説明文中の角カッコ ([]) は、キーボード上のキーやメニュー項目，ダイアログ ボックスの名前などを表します。 Visual Basic for MS-DOS のマニュアルやオンライン ヘルプでは、キーの表記にジェネリック表記を採用しています。一部のキーは、キーボード上に書かれた表記と異なりますから注意してください。例えばリターン キーは [ENTER]，メニューをアクティブにするためのキーは [メニュー] キーです。 Visual Basic for MS-DOS でのジェネリック表記は、Microsoft Windows 用のアプリケーションと合わせているため、 Microsoft Basic や QuickBasic での表記とは異なります。 ジェネリック表記と実際のキーボード上の表記との対応は、『機種依存情報』またはオンライン ヘルプを参照してください。
[メニュー] + [F1]	プラス記号 (+) は、キーの組み合わせを表します。例えば [メニュー] + [F1] は、[メニュー] キーを押しながら [F1] キーを押すことを表します。
Multiple Document Interface (MDI)	略語はその用語を最初に使うときは正式名称を書きます。

キーボード上のキーの表記方法 (PC-9801 シリーズ)

Visual Basic for MS-DOS のマニュアルおよびオンライン ヘルプでは、キーボード上の各キーを示すために、キーボードに実際に表示されている表記ではなく、ジェネリック (標準) 表記を採用しています。これは、Microsoft Excel や Microsoft Word for Windows のマニュアルで使われているものと基本的に同じです。PC-9801 シリーズにおける実際のキー表記とジェネリック表記の対応は、次のようになっています。

ジェネリック表記	実際のキー表記
[ESC]	[ESC]
[TAB]	[TAB]
[CTRL]	[CTRL]
[CAPS LOCK]	[CAPS]
[NUM LOCK]	
[SCROLL LOCK]	
[PAUSE]	
[SHIFT]	[SHIFT]
[英数カナ]	[ｶﾅ]
[ﾒﾆｭｰ]	[GRPH]
[無変換]	[NFER]
[SPACE]	(スペース)
[変換]	[XFER]
[漢字]	[CTRL]+[XFER]
[ENTER]	(リターン)
[BS]	[BS]
[INS]	[INS]
[DEL]	[DEL]
[HOME]	[HOME CLR]
[END]	[HELP]
[PAGE DOWN]	[ROLL UP]
[PAGE UP]	[ROLL DOWN]
[STOP]	[STOP]
[COPY]	[COPY]

ジェネリック表記	実際のキー表記
[↑]	[↑]
[↓]	[↓]
[←]	[←]
[→]	[→]
[F1]	[f・1]
[F2]	[f・2]
[F3]	[f・3]
[F4]	[f・4]
[F5]	[f・5]
[F6]	[f・6]
[F7]	[f・7]
[F8]	[f・8]
[F9]	[f・9]
[F10]	[f・10]
[F11]	[vf・11], [GRPH]+[f・1]
[F12]	[vf・12], [GRPH]+[f・2]
[F13]	[vf・13], [GRPH]+[f・3]
[F14]	[vf・14], [GRPH]+[f・4]
[F15]	[vf・15], [GRPH]+[f・5]
[F16]	

1 PC-9801 シリーズ固有の仕様

対応するハードウェアによって Visual Basic for MS-DOS にはいくつかの種類があり，使用する機種に依存していくつかの機能に違いがあります。ここでは、PC-9801シリーズに依存する仕様についてまとめて説明します。

機種に依存する機能は、主に次の 3 種類に分類できます。

- **画面表示に関するもの**

  画面解像度，同時表示可能色数，スクリーンページの枚数など、スクリーン モードに関するもの

- **入力に関するもの**

  キーボード スキャンコードなど

- **その他**

  コントロール コードに対応する表示文字など

1.1 画面表示とスクリーン モード

画面への表示方法を規定しているモードを、スクリーン モードと呼びます。スクリーン モードによって、フォームが表示できるかどうか、グラフィックスが表示できるかどうか、テキストがどのように表示されるか、解像度や同時表示可能な色数などが決まります。スクリーン モードには、大きく分けて次の 3 種類があります。

- **テキスト モード**

  アプリケーションからはテキスト画面は操作できますが、グラフィックスの機能は利用できません。フォームが表示できます。

- **グラフィックス モード**

  アプリケーションからはグラフィックス画面だけを操作できます。テキストも、グラフィックス画面に表示されます。グラフィックス モードでは、フォームは表示できません。

- **スーパーインポーズ モード**

  アプリケーションからはテキスト画面とグラフィックス画面の両方を操作できます。テキストはテキスト画面に表示され、グラフィックスはグラフィックス画面に表示されます。フォームが表示されている状態では、グラフィックス ステートメントは実行できません。

SCREEN ステートメントによるスクリーン モードの設定

スクリーン モードを設定するには、**SCREEN** ステートメントを使います。**SCREEN** ステートメントの構文は次のとおりです。

SCREEN *mode%*[, [*paletteswitch%*][, [*apage%*][, [*vpage%*][, *noinit%*]]]]

注　意　フォームが表示されている状態では、**SCREEN** ステートメントは実行できません。フォームが表示されているときに **SCREEN** ステートメントを実行すると、“フォームが表示されているときは無効です” エラーが発生します。**SCREEN.HIDE** を実行して、必ずすべてのフォームを非表示にしてください。

PC-9801 シリーズでは、次の 5 種類のスクリーン モードが用意されています。各引数に指定できる値は次のとおりです。“－” は指定できないか、意味を持たないことを示します。

mode%	*paletteswitch%*	*apage%/vpage%*	解像度	画面のモード
0, 81	－	－	－	テキスト
82	－	－	－	テキスト
1, 87	0～3	0～1	640×400	グラフィックス
2, 88	0～3	0～1	640×400	スーパーインポーズ
84	0～3	0～3	640×200	スーパーインポーズ

注　意　プログラミング環境上でアプリケーションを実行する場合、*apage%* および *vpage%* に指定できる値に制限があります。くわしくは「グラフィックス スクリーン ページ」を参照してください。

スクリーン モード 0 がテキスト モード，1 がグラフィックス モード，2 がスーパーインポーズ モードを示します。PC-9801 シリーズでは、0 は 81、1 は 87、2 は 88 と同じ意味です。82, 84 と合わせ、スクリーン モードには 5 種類があります。

テキスト モードには、0 (81) と 82 の 2 種類がありますが、フォームの表示方法が異なります。0 (81) は、すべての画面表示にテキスト画面だけを使い、フォームもテキスト画面に表示されます。82 では、フォームはグラフィックス画面とテキスト画面の両方を使って表示されます。ただしこの場合でも、グラフィックス画面の管理は Visual Basic for MS-DOS が行い、作成するアプリケーションでグラフィックス ステートメントを実行することはできません。

デフォルトのスクリーン モード、つまりアプリケーション中で特に指定しないときのスクリーン モードは、フォームを使うアプリケーションでは 82、フォームを使わないアプリケーションでは 0 (81) です。どちらのスクリーン モードに設定するかは、最初に画面に対する表示を行うときに、Visual Basic for MS-DOS が自動的に判断します。

スクリーン モード 81, 2 (88), 84 では、フォームはテキスト画面だけを利用して表示されます。この場合、コマンド ボタンやピクチャー ボックスなどは影を付けずに表示され、**SCREEN** オブジェクトの ControlPanel プロパティの 3D の指定は無視されます。スーパーインポーズ モードであるスクリーン モード 2 (88), 84 の場合でも、フォームが表示されている状態ではグラフィックス命令は実行できないことに注意してください。フォームがもっともきれいに表示できるのは、スクリーン モード 82 です。フォームを使うアプリケーションでは、通常このモードを使います。

グラフィックス モードであるスクリーン モード 1 (87) では、フォームは表示できません。

デフォルト (スクリーン モード 0 (81) または 82) はテキスト モードであるため、グラフィックス ステートメントを使うには、必ず **SCREEN** ステートメントを実行して、グラフィックスまたはスーパーインポーズ モードに設定する必要があります。**SCREEN** ステートメントは、フォームが表示されている状態では実行できないことに注意してください。実行するには、**SCREEN.HIDE** ですべてのフォームを非表示にします。

MS-DOS の画面は、スクリーン モード 0 (81) と同じです。フォームやグラフィックスを使わないユーティリティ プログラムを作成するときは、スクリーン モード 0 (81) に設定してください。また、このようなプログラムの出力結果を画面に表示したまま終了したいときは、noinit% 引数に 1 を指定してください。

Microsoft Basic バージョン 7.1 でサポートされていたスクリーン モード 89 と、QuickBasic バージョン 4.5 でサポートされていた環境変数 QBGRPNEC は廃止されています。また、スクリーン モード 82 の意味が異なっていることに注意してください。

Microsoft Basic バージョン 7.1 では、Visual Basic for MS-DOS と同様にデフォルトはテキストモードでしたが、QuickBasic バージョン 4.5 ではスーパーインポーズ モード (88 と同じ) であったことに注意してください。

色値, パレット番号, 色番号

色を示す値には、色値, パレット番号, 色番号の 3 種類があります。

色値

画面上の色は、R (赤), G (緑), B (青) の 3 原色の組み合わせによって表されていますが、**SCREEN** ステートメントの *paletteswitch%* 引数に指定する値によって、RGB それぞれに何ビットを使うかが決まります。つまりこの引数に指定する値によって、何色中何色を使うかが決まります。次のようになります。

paletteswitch %	表示可能な色数	1 原色当たりのビット数	パレット番号の有効範囲	色値の有効範囲
0	(同時表示可能な最大色数)		0～(同時表示可能な最大色数)	
1	16 色中 16 色	2	0～15	0～15
2	64 色中 16 色	3	0～15	0～63
3	4096 色中 16 色	4	0～15	0～4095

paletteswitch% 引数のデフォルトの値は 0 です。0 は、そのハードウェアで同時表示可能な最大色数を使うことを示します。PC-9801 シリーズでは、0 は 1 (16 色中 16 色表示) と同じ意味です。

Basic では、グラフィックスで使う色を指定するときに色値とパレット番号という 2 つの値を使いますが、パレット番号が論理的な値であるのに対し、色値は RGB の物理的な組み合わせを示します。実際には、グラフィックス パレットでパレット番号と色値が結び付けられており、図形の描画を行うグラフィックス ステートメントでパレット番号を指定すると、対応する色値が示す色が使われます。通常、色値を指定するのは、**PALETTE** ステートメントでパレットの設定を変更するときだけです。

パレット番号および色値としてどのような値が有効かは、**SCREEN** ステートメントの *paletteswitch%* 引数の値によって決まります。上の表を参照してください。

色値の指定方法は、**SCREEN** ステートメントの *mode%* 引数で指定したスクリーン モードと、*paletteswitch%* 引数の値によって決まります。例えば *paletteswitch%* 引数に 3 を指定して 4096 色中 16 色表示を選択した場合は、1 原色当たり 4 ビットの計 12 ビットで色を表します。指定する順番は、RGB (赤緑青) の順です。例えばもっとも明るい水色 (シアン) は、次のような色値で表されます。

R ビット	G ビット	B ビット	色値
0000	1111	1111	255 (=&H0FF)

パレット番号

パレット番号は、グラフィックス ステートメントで色を指定するときに使う値です。ディスプレイに表示される色は、実際には R (赤), G (緑), B (青) それぞれの輝度によって決まります。この輝度をビット単位で表した値を色値と呼びます。色値の指定方法はハードウェアの設定やスクリーン モードによって異なるため、グラフィック ステートメントを実行するたびに色値を指定していたのでは不便です。そこでパレットと呼ばれるテーブルを作り、物理的に色を示す値である色値を、論理的な値であるパレット番号に結び付けています。

パレット番号の最小値は 0 です。グラフィックスが表示できるスクリーン モードでは、パレット番号 0 に設定されている色が、画面全体の背景色になります。

パレット番号の最大値は、そのときのハードウェア構成における同時表示可能な最大色数です。PC-9801 シリーズでは最大 16 色の同時表示が可能であり、パレット番号の最大値は 15 になります。デフォルトでは、パレット番号の最大値に設定されている色が、カレントのグラフィックスの色になります。

グラフィックス パレットのデフォルトの設定は次のようになっています。

パレット番号	表示される色	パレット番号	表示される色
0	黒	8	灰
1	青	9	青 (明)
2	緑	10	緑 (明)
3	水	11	水 (明)
4	赤	12	赤 (明)
5	紫	13	紫 (明)
6	黄	14	黄 (明)
7	白	15	白 (明)

パレット番号に対する色の設定は、**PALETTE** ステートメントで行います。

テキスト モード (0 (81), 82) にはパレットはありません。

色番号

色番号は、フォームを使う場合の画面各部の色の指定と、フォームを使わない場合の文字色や文字の背景色の指定に使います。パレット番号が、**PALETTE** ステートメントによって番号とそれが示す色の関係を変化させることができるのに対し、色番号は表示色と直接結び付けられており、プログラム中から色と番号の関係を変化させることはできません。

フォームを使う場合の色の指定は、**SCREEN** オブジェクトの ControlPanel プロパティ、およびフォームや各コントロールの BackColor, ForeColor プロパティで設定します。また、フォームを使わない場合の文字色や文字の背景色の設定は、**COLOR** ステートメントで行います。

色番号は 0～15 の値で、デフォルトのパレット番号の設定と同じ、16 種類の色を示します。

ただし PC-9801 シリーズのテキスト画面では、表示できる色数は 8 色です。この場合、1～7 と 9～15 は同じ色の繰り返しになり、8 は 0 と同じ色になります。テキスト画面における色番号は、デフォルトのパレット番号の設定と、次のような関係になっています。

色番号	表示される色	デフォルトで同じ色の パレット番号
0, 8	黒	0
1, 9	青 (明)	9
2, 10	緑 (明)	10
3, 11	水 (明)	11
4, 12	赤 (明)	12
5, 13	紫 (明)	13
6, 14	黄 (明)	14
7, 15	白 (明)	15

フォームを使うかどうかにかかわらず、スクリーン モード 0 (81), 82, 2 (88), 84 の文字色と、スクリーン モード 0 (81), 2 (88), 84 の文字の背景色は、テキスト画面を使って表示されます。

色の指定方法 (COLOR ステートメント)

COLOR ステートメントを使うと、フォームを使わない場合の文字や文字の背景色、およびグラフィックス ステートメントで図形を描くときのデフォルトの色を設定できます。**COLOR** ステートメントは、フォームが表示されている状態では利用できません。**COLOR** ステートメントの構文は次のとおりです。

COLOR [*foreground*][, [*background*]]

COLOR ステートメントの機能はそのときのスクリーン モードによって異なり、次のようになっています。

スクリーン モード 82 (テキスト モード) の設定

- 文字の色と、文字の背景色を設定します。このスクリーン モードでは、文字はテキスト画面、文字の背景色はグラフィックス画面に表示されます。ただし、ユーザーはグラフィックス画面を直接操作することはできず、文字の背景色はグラフィックス パレットの影響を受けません。
- *foreground* には、文字の色を示す 0～31 の値を指定します。0～15 は色番号を示します。ただし実際に表示できる色は 8 色です。16～31 は、0～15 と同じ色で点滅します。
- *background* には、文字の背景色を示す 0～15 の色番号を指定します。背景色は 16 色から選択できます。デフォルトの文字色は 0 (黒) です。

スクリーン モード 0 (81) (テキスト モード) の設定

- 文字の色と、文字の背景色を設定します。このスクリーン モードでは、文字と文字の背景色ともにテキスト画面だけを使って表示されます。どちらも 8 色から選択できますが、文字色または背景色のどちらかは 0 (黒) でなければなりません。
- *foreground* には、文字の色を示す 0～31 の値を指定します。0～15 は色番号を示します。ただし実際に表示できる色は 8 色です。16～31 は、0～15 と同じ色で点滅します。*foreground* は、*background* が 0 のときだけ意味を持ちます。*background* が 0 以外のときに *foreground* を指定しても無視され、0 と見なされます。
- *background* には、文字の背景色を示す 0～15 の色番号を指定します。ただし実際に表示できる色は 8 色です。

スクリーン モード 1 (87) (グラフィックス モード) の設定

- グラフィックスのデフォルトの色と、文字の背景色を設定します。このスクリーン モードでは、文字もグラフィックス画面を使って表示され、グラフィックスのデフォルトの色の影響を受けます。

- *foreground* には、0～15 のパレット番号を指定します。このパレット番号に設定されている色がグラフィックスを描画するときのデフォルトの色になり、描画ステートメントで特に色を指定しないと、この色で図形が描かれます。また、文字を表示するときもこの色が使われます。
- *background* には、文字の背景色を示す、0～15 のパレット番号を指定します。この色が、文字を出力するときの背景色になります。スクリーン モード 0 (81) と異なり、文字か文字の背景色のどちらかが黒でなければならないという制限はありません。
- QuickBasic や Microsoft Basic では、グラフィックス モードで **COLOR** ステートメントの *background* 引数を指定すると、パレット番号 0 の設定が変更され、画面全体の背景色が変わっていました。Visual Basic for MS-DOS のグラフィックス モードで画面全体の背景色を変えるには、**PALETTE** ステートメントを使って、パレット番号 0 の設定を変更するようにしてください。

スクリーン モード 2 (88), 84 (スーパーインポーズ モード) の設定

- 文字色, グラフィックスのデフォルトの色, 文字の背景色を設定します。このスクリーン モードでは、スクリーン モード 81 と同様に、文字と文字の背景色ともにテキスト画面だけを使って表示されます。
- *foreground* に指定する値は、文字色に関してはスクリーン モード 81 と同じ意味で、グラフィックスのデフォルトの色に関してはスクリーン モード 87 と同じ意味です。この値は、文字色に対しては色番号を示し、グラフィックスのデフォルトの色に対してはパレット番号を示します。したがって、パレットの設定が変更されている状態では、同じ値を指定しても、文字色とグラフィックスで違う色で表示される場合があります。
- *background* は、スクリーン モード 81 の場合と同じ意味になります。

グラフィックス スクリーン ページ

グラフィックス モードおよびスーパーインポーズ モードでは、グラフィックス画面を操作できます。グラフィックス画面は、グラフィックス スクリーン ページと呼ばれる何枚かのページに分けられており、ページを切り換えることにより、グラフィックス画面をすばやく書き換えるように見せることができます。グラフィックス スクリーン ページの枚数はスクリーン モードによって決まり、PC-9801 シリーズでは、スクリーン モード 1 (87), 2 (88) のときは 2 枚 (ページ 0 および 1)、スクリーン モード 84 のときは 4 枚 (ページ 0～3) です。**SCREEN** ステートメントの *apage%* 引数は、グラフィックス ステートメントによる操作の対象となるページを指定し、*vpage%* は表示するページを指定します。

プログラミング環境 (VBDOS.EXE) を /F スイッチを指定せずに起動すると、スクリーン モード 1 (87) または 2 (88) のときのページ 1, スクリーン モード 84 のときのページ 2 および 3 に相当する部分は、プログラミング環境が使用します。この場合、作成するアプリケーションで、*apage%* または *vpage%* 引数にこれらのページを指定すると、“引数が有効範囲にありません” エラーが発生します。また、**PCOPY** ステートメントでフォームの表示に使われているページを指定すると、同じエラーが発生します。

これらのページを操作するアプリケーションを作成する場合は、/F スイッチを指定してプログラミング環境 (VBDOS.EXE) を起動するようにしてください。

PCOPY ステートメント

PCOPY ステートメントは、次のような構文で引数にスクリーン ページの番号を指定します。

PCOPY *sourcepage%, destinationpage%*

指定できる値の範囲は、**SCREEN** ステートメントの *apage%* および *vpage%* 引数に指定できるものと同じです。

スクリーン モード 2 (88) では、2 枚目のグラフィックス プレーンを、またスクリーン モード 84 では 3 枚目と 4 枚目のグラフィックス プレーンをフォームの表示に使います。この状態でも **PCOPY** ステートメントは実行できますが、意味を持ちません。

画面行数とフォーム，INPUTBOX$，MSGBOX の表示

画面の行数を設定するには、**WIDTH** ステートメントを使います。**WIDTH** ステートメントの構文は次のとおりです。

WIDTH {*screenwidth%* | , *screenheight%* | *screenwidth%, screenheight%*}

*screenwidth%*には 80、*screenheight%* には 20 または 25 が指定できます。

画面表示行数が 20 行のときは、フォームは表示できません。また、**INPUTBOX$** 関数，**MSGBOX** 関数，**MSGBOX** ステートメントも実行できません。実行しようとすると、“スクリーン モードが不正です” エラーが発生します。ただし、アプリケーションの実行開始時に Visual Basic for MS-DOS がデフォルトのスクリーン モードを決める際に、フォームを使うアプリケーションに対しては、スクリーン モードを 82 に設定すると同時に、画面の表示行数を 25 行に初期化するため、通常は画面行数を意識する必要はありません。

テキスト画面のみを使うときの制限

マウス ポインタ

プログラミング環境上でのマウス ポインタの形状は、起動時に /F オプションが指定されているかどうかによって異なります。/F オプションを指定した場合、MS-DOS のコマンドラインに表示されるような四角になります。

アプリケーション実行時のマウス ポインタの形状は、フォームがグラフィックス画面も使って表示されているか、テキスト画面だけを使って表示されているかによって異なります。

ControlPanel プロパティの 3D

SCREEN 特殊オブジェクトの ControlPanel プロパティの要素 15 は、コントロールを 3D 表示するかどうかを制御しています。デフォルトの設定は True (-1) で、コントロールは立体的に見えるように表示されます。ただしこの設定にかかわらず、プログラミ

ング環境およびフォーム デザイナを /F オプションを指定して起動したときと、スクリーン モード 0 (81), 2 (88), 84 でテキスト画面にフォームを表示するときは、コントロールは立体的に表示されません。

1.2 キーボードからの入力

キーボードのキー配置やキートップに示されているキーの名称は、機種によって異なります。

キーの名称

Visual Basic for MS-DOS のマニュアルおよびオンライン ヘルプでは、それぞれのキーを示すために、機種に依存しないジェネリック表記を使っています。実際にキートップに示されているキーの名称とジェネリック表記との関係は、「マニュアルの表記規則」またはオンライン ヘルプを参照してください。例えば [GRPH] キーは [ﾒﾆｭｰ], [ROLL UP] キーは [PAGE DOWN] キーと表記されます。

KeyDown, KeyUp イベントの *KeyCode* 引数

KeyDown, KeyUp イベントは、ユーザーがキーを押したり離したりするときに発生しますが、これらのイベントに対するイベント プロシージャには、押されたキーを示す *KeyCode* 引数が渡されます。各キーと、*KeyCode* 引数として渡される値の対応は次のとおりです (これらの値はハードウェアに依存するため、ジェネリック表記ではなく実際のキートップの表記どおりに示してあります)。

キー	16 進	10 進
[BS]	08	8
[TAB]	09	9
(リターン)	0D	13
[SHIFT]	10	16
[CTRL]	11	17
[GRPH]([ﾒﾆｭｰ])	12	18
[ESC]	1B	27
[XFER]	1C	28
[NFER]	1D	29
(スペース)	20	32
[ROLL DOWN]	21	33
[ROLL UP]	22	34
[HELP]	23	35
[HOME]	24	36
[←]	25	37
[↑]	26	38
[→]	27	39

キー	16 進	10 進
[↓]	28	40
[INS]	2D	45
[DEL]	2E	46
[0]	30	48
[1]	31	49
[2]	32	50
[3]	33	51
[4]	34	52
[5]	35	53
[6]	36	54
[7]	37	55
[8]	38	56
[9]	39	57
[A]	41	65
[B]	42	66
[C]	43	67
[D]	44	68

キー	16 進	10 進
[E]	45	69
[F]	46	70
[G]	47	71
[H]	48	72
[I]	49	73
[J]	4A	74
[K]	4B	75
[L]	4C	76
[M]	4D	77
[N]	4E	78
[O]	4F	89
[P]	50	80
[Q]	51	81
[R]	52	82
[S]	53	83
[T]	54	84
[U]	55	85
[V]	56	86
[W]	57	87
[X]	58	88
[Y]	59	89
[Z]	5A	90
[:]	BA	186
[;]	BB	187
[,]	BC	188
[-]	BD	189
[.]	BE	190
[/]	BF	191
[@]	C0	192
[[]	DB	219
[¥]	DC	220
[]]	DD	221
[^]	DE	222

* はテンキー上のキーを示します。

キー	16 進	10 進
[f・1]	70	112
[f・2]	71	113
[f・3]	72	114
[f・4]	73	115
[f・5]	74	116
[f・6]	75	117
[f・7]	76	118
[f・8]	77	119
[f・9]	78	120
[f・10]	79	121
[vf・1]	7A	122
[vf・2]	7B	123
[vf・3]	7C	124
[vf・4]	7D	125
[vf・5]	7E	126
[0]*	60	96
[1]*	61	97
[2]*	62	98
[3]*	63	99
[4]*	64	100
[5]*	65	101
[6]*	66	102
[7]*	67	103
[8]*	68	104
[9]*	69	105
[*]*	6A	106
[+]*	6B	107
[,]*	6C	108
[-]*	6D	109
[.]*	6E	110
[/]*	6F	111
[=]*	92	146

KEY ステートメントの *keyboardflag%* と *code%* 引数

KEY ステートメントでユーザー定義キーのトラップを設定するとき、次のような構文を使います。

KEY *n%*, **CHR$**(*keyboradflag%*) + **CHR$**(*code%*)

keyboradflag% 引数に指定できる値は次のとおりです。

キー	*keyboardflag%* に指定する値	ビット位置
[SHIFT]	1	0
[CTRL]	2	1
[メニュー]	4	2
キーの組み合わせを無視	128	7

[SHIFT], [CTRL], [メニュー] キーの指定方法は、MouseDown, MouseUp, KeyDown, KeyUp イベントなどの *Shift* 引数と同じで、ビットフィールドになっています。詳しくはリファレンスまたはオンライン ヘルプ、および『プログラミング ガイド』の第 13 章を参照してください。

keyboradflag% 引数に 128 を加えると、[SHIFT], [CTRL], [メニュー] の各キーが押されているかどうかをチェックしないようになります。これによって、どのようなキーの組み合わせが押されても、単独のキーが押されたときと同じように扱うことができます。

code% 引数に指定できる値は、以下のキーを除き、KeyDown, KeyUp イベントの *KeyCode* 引数と同じです。

キー	16 進	10 進
[STOP]	03	3
[COPY]	2A	42
[_] (最前列右下のキー)	DF	223

INKEY$ 関数が返す値

INKEY$ 関数はキーボードから入力された値を返しますが、返される値には次の 3 種類があります。

- 通常のキーが押された場合： ASCII コード
- 2 バイト文字が入力された場合： 漢字連続コード
- 特殊キーが押された場合： 特殊キー コード

漢字連続コードは、JIS 漢字コード (またはシフト JIS コード) の先頭の文字である 2 バイトのスペースを 10 進数の 256 として、JIS 漢字コード順にすきまなく並べて行ったコードです。各文字の漢字連続コードは、**ASC** 関数で求めることができます。

**INKEY$** 関数に対して特殊キーが押されたとき、次のようなコードが返されます。表中の数値はすべて 16 進数を表し、“00+*nn*” は CHR$(0)+CHR$(&H*nn*) を示しています。[CAPS LOCK] キーや [英数カナ] キーは、戻される値に影響しません。また、[メニュー] キーも影響しません。[CTRL], [SHIFT] が組み合わせて押されている場合は、優先順位は [CTRL], [SHIFT] の順です。

	単独	[SHIFT]+	[CTRL]+
[BS]	08	08*	08*
[TAB]	09	09*	09*
[ESC]	1B	1B*	1B*
(リターン)	0D	0D*	0D*
(スペース)	20	20*	20*
[f·1]	00+3B	00+54	00+5E
[f·2]	00+3C	00+55	00+5F
[f·3]	00+3D	00+56	00+60
[f·4]	00+3E	00+57	00+61
[f·5]	00+3F	00+58	00+62
[f·6]	00+40	00+59	00+63
[f·7]	00+41	00+5A	00+64
[f·8]	00+42	00+5B	00+65
[f·9]	00+43	00+5C	00+66
[f·10]	00+44	00+5D	00+67
[vf·1]	00+85	00+87	00+89
[vf·2]	00+86	00+88	00+8A
[vf·3]	00+D4	00+D7	00+DA
[vf·4]	00+D5	00+D8	00+DB
[vf·5]	00+D6	00+D9	00+DC
[ROLLUP]	00+51	00+51*	00+51*
[ROLLDOWN]	00+49	00+49*	00+49*
[INS]	00+52	00+52*	00+52*
[DEL]	00+53	00+53*	00+53*
[↑]	00+48	00+48*	00+48*
[→]	00+4B	00+4B*	00+4B*
[←]	00+4D	00+4D*	00+4D*
[↓]	00+50	00+50*	00+50*
[HOME/CLR]	00+47	00+77	—
[HELP]	00+4F	00+4F*	00+4F*
[STOP]	—	—	—
[COPY]	—	—	—

* これらのキーは、単独で押されたときと同じ値を返します。

[CTRL]+[HOME/CLR] は何も押されていないとみなされ、戻り値は null 文字列になります。このキーの組み合わせを感知することはできません。

プログラミング環境で実行している場合や /D オプションを指定してコンパイルしたプログラムでは、**INKEY$** に対して [STOP] キーが入力されると、プログラムの実行は終了します。また、[CTRL]+[C] は、あらゆる場合で他のキーと同様です。

ファンクション キー [F16]

フォーム デザイナのメニュー デザイン ウィンドウでのショートカット キーの設定や、**KEY** ステートメントを使ったファンクション キーに対する文字列の設定などでは、ファンクション キーとして [F16] が設定可能です。ただし、実際に対応するキーがないため、設定しても意味がありません。

1.3 その他

ASCII コード, 漢字連続コード以外のコードに対応する文字

Visual Basic for MS-DOS では、ASCII コードや漢字連続コードで文字が定義されていないコード (&H0～&H1F, &H7F, &HFD～&HFF) に対して、罫線や記号などの文字を割り当てています。例えば MS-DOS では、画面に対して 16 進数の 7 を出力すると、ビープ音が鳴ります。これに対して Visual Basic for MS-DOS では、次のステートメントを実行すると、画面には罫線が出力されます。

```
PRINT CHR$(7)
```

コードと、それに割り当てられている文字の関係は次のとおりです。

16 進	10 進	出力される文字	
00	0	(null)	
01	1	細線　垂直	│
02	2	細線　水平	─
03	3	細線　左上角	┌
04	4	細線　右上角	┐
05	5	細線　左下角	└
06	6	細線　右下角	┘
07	7	細線　左辺	├
08	8	細線　右辺	┤
09	9	(TAB)	
0A	10	(LF)	
0B	11	細線　上辺	┬
0C	12	細線　下辺	┴
0D	13	(CR)	
0E	14	細線　十文字	┼
0F	15	太線　垂直	┃

10	16	太線　水平	━
11	17	太線　左上角	┏
12	18	太線　右上角	┓
13	19	太線　左下角	┗
14	20	太線　右下角	┛
15	21	太線　左辺	┣
16	22	太線　右辺	┫
17	23	太線　上辺	┳
18	24	太線　下辺	┻
19	25	太線　十文字	╋
1A	26	(EOF)	
1B	27	コントロール ボックス	-
1C	28	左矢印	←
1D	29	右矢印	→
1E	30	上矢印	↑
1F	31	下矢印	↓
7F	127	スクロールバー	·
FD	253	最大化ボタン	↑
FE	254	最小化ボタン	↓
FF	255	復元ボタン	◆

これらのコードと文字の関係は、次のステートメントで有効です。

- **PRINT** ステートメントと **WRITE** ステートメント
- **PRINT** メソッド
- **OPEN** "SCRN:" でオープンした画面に対する **PRINT #, WRITE #** ステートメント

次のステートメントでは、MS-DOS 上と同じように扱われ、上記の関係は影響しません。

- **LPRINT** ステートメント
- **OPEN** "CONS:" でオープンした画面に対する **PRINT #, WRITE #** ステートメント

注　意　これらの罫線文字をダブルクォーテーションで囲んで、リテラル文字列として **ASC** 関数の引数に指定することは避けてください。

タイマーの精度

タイマー コントロールの Interval プロパティは、Timer イベントが発生する時間間隔をミリ秒単位で指定します。また、従来の Basic から用意されている **ON TIMER, TIMER** ステートメントによるタイマー割り込み (イベント トラップ) では、秒単位で経過時間を設定します。ただし実際の計測の精度はハードウェアに依存し、必ずしも指定した時間間隔ごとに正確に制御が分岐するわけではありません。また、そのとき行われている別の処理がどのようなものであるかによって、計測の精度は大きく異なります。指定する時間間隔は、標準的な時間間隔とお考えください。

UTILITY.BAS サンプル アプリケーション

UTILITY.BAS は、PC-9801 シリーズ固有の機能を集めたサンプル アプリケーションです。次の 3 つのプロシージャが含まれています。

KPut SUB プロシージャ

機能

グラフィックス画面に対して、指定したサイズで文字を表示します。

構文

KPut *string-expression$, fontsize%, x%, y%, fcolor%*

string-expression$	画面に表示するテキスト。
fontsize%	表示する文字の大きさを示す整数。
x%	出力する文字の左上の x 座標を示す整数。
y%	出力する文字の左上の y 座標を示す整数。
fcolor%	表示するテキストのパレット番号を示す 0～15 の値。

解説

string-expression$ で指定した文字列を、*fontsize%* が示す大きさでグラフィックス画面に出力します。表示が終わると、グラフィックス出力位置はテキストの右上の位置に移動します。

PrnOut SUB プロシージャ

機能

プリンタに特殊文字を出力します。

構文

PrnOut *string-expression$*

string-expression$	画面に表示するテキスト。

解説

LPRINT ステートメントでプリンタにコードを送る場合、&H81～&HA0, &HE0～&HFE までのコードを送ることができません。このプロシージャを使うと、これらのコードもプリンタに出力できます。

Hcopy SUB プロシージャ

機能

PC-PR201 系のプリンタに、画面のハードコピーを出力します。

構文

Hcopy *mode%*

mode%	出力のモードを示す 0～3 の整数。 0　テキスト画面のみ 1　テキスト画面とグラフィックス画面 (16階調) 2　テキスト画面とグラフィックス画面 (8階調) 3　テキスト画面とグラフィックス画面 (16階調反転)

解説

PC-PR201 系以外のプリンタでは正常に動作しません。

2 グラフィックス

Visual Basic for MS-DOS のグラフィックスを利用すると、アプリケーションに視覚的な効果を持たせることができます。ここでは、グラフィックスのためのステートメントと関数を使って、画面にさまざまな形，色，パターンを描画する方法を説明します。

注　意　フォームが表示されている状態では、グラフィックス ステートメントや **SCREEN** ステートメントは実行できません。フォームが表示されているときに実行すると、“フォームが表示されているときは無効です”エラーが発生します。

2.1 スクリーン モードの設定

グラフィックスを利用するには、**SCREEN** ステートメントを使って、グラフィックスが表示できるスクリーン モードに設定する必要があります。**SCREEN** ステートメントの構文は、次のとおりです。

SCREEN *mode%*, [*paletteswitch%*][, [*apage%*][, [*vpage%*][, *noinit%*]]]]

フォームが表示されている状態では、**SCREEN** ステートメントは実行できません。**SCREEN** ステートメントの使い方やスクリーン モードについて詳しくは、第 1 章「PC-9801 シリーズ固有の仕様」またはオンライン ヘルプを参照してください。

1 つのアプリケーションでフォームとグラフィックスの両方を扱いたいときは、すべてのフォームを非表示にしてから、スクリーン モードを切り換える必要があります。例を次に示します。

```
SCREEN.HIDE        'すべてのフォームを非表示にします.
SCREEN mode%       'グラフィックス モードに切り換えて，グラフィックスを表示
                   'します.
   .
   .
   .
SCREEN 82          'テキスト モードに戻します.
SCREEN.SHOW        'あらためて，フォームを表示します.
```

2.2 ピクセルと画面座標

ディスプレイに表示される図形は、“ピクセル”という点から構成されています。グラフィック画面上に規則正しく縦横にならんだ各ピクセルに色を設定することによって、図形を表示することができます。このグラフィック画面上のピクセル数は、ハードウェアの構成や **SCREEN** ステートメントで設定したスクリーン モードによって変化しますが、ピクセル数が多いほど解像度が高くなり、鮮明な図形が表示できます。

グラフィックスが表示できるスクリーン モードに設定されていると、1 つのピクセルは座標 (x, y) によって指定することができます。x 座標は画面の左端からのピクセル数を表し、y 座標は通常、画面の上端からのピクセル数を表します。

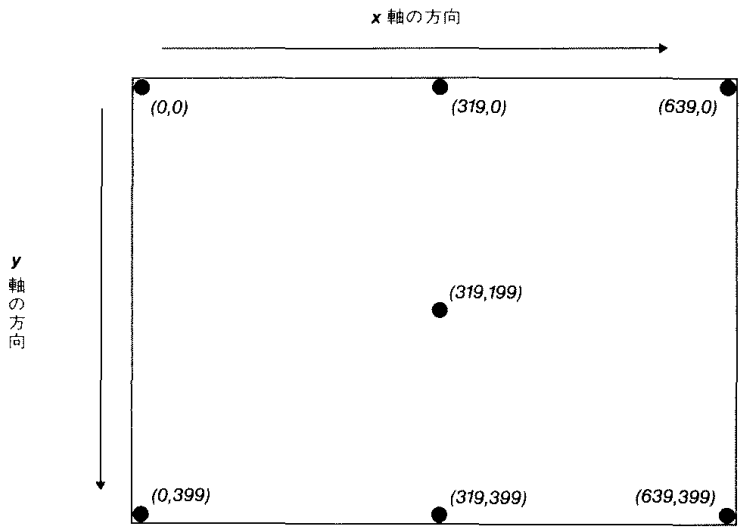


図 2.1　ピクセルと物理座標

Visual Basic for MS-DOS では、画面上の座標を使って、グラフィックスの表示位置 (直線の両端や円の中心など) を指定します。座標については「点，線，ボックスなどの基本図形の描画」で説明します。

グラフィックス画面の座標は、**LOCATE** ステートメントなどで指定するテキスト画面の座標とは異なります。テキスト画面では、座標は文字単位で文字の位置を表し、次のように行と桁を指定します。

```
'13 行、15 桁に移動して，下のメッセージを表示します．
LOCATE 13, 15
PRINT "このメッセージは画面の中央に表示されます．"
```

グラフィックス画面では座標を (横、縦) の順に指定するのに対して、テキスト画面では (縦、横) の順に指定することに注意してください。また、原点の座標は、グラフィックス画面では (0,0) ですが、テキスト画面では (1,1) です。**LOCATE** ステートメントは、グラフィックス画面に対する出力には影響しません。

2.3 点，線，ボックスなどの基本図形の描画

ここでは、Visual Basic for MS-DOS のグラフィックス ステートメントを使って、簡単な図形を描く方法を説明します。

PSET と PRESET による点の描画

グラフィックス出力のもっとも基本的な操作は、各ピクセルの ON と OFF です。この操作には、**PSET** ステートメント (ピクセルのセット) と **PRESET** ステートメント (ピクセルのリセット) を使います。**PSET** (*x!*, *y!*) ステートメントは、座標 (*x!*, *y!*) のピクセルに点を描きます。また、**PRESET** (*x!*, *y!*) ステートメントは、座標 (*x!*, *y!*) のピクセルの点を消去します。

この構文を、次に示します。

PSET (*x!*, *y!*), *color%*
PRESET (*x!*, *y!*), *color%*

色の指定方法については、「色の選択」を参照してください。

特に色を指定しなければ、**PSET** ステートメントは現在のグラフィックスの色 (前景色) で点を描き、**PRESET** ステートメントは現在の背景色で描きます。このため、**PRESET** ステートメントで引数 *color* を省略すると、**PSET** ステートメントで描いた点を消去することができます。次に例を示します。

```
PRINT "何かのキーを押すと終了します."

DO
    '左から右に水平線を描きます.
    FOR X% = 0 TO 639
        PSET (X%, 200)
    NEXT
    '右から左に線を消していきます.
    FOR X% = 639 TO 0 STEP -1
        PRESET (X%, 200)
    NEXT

LOOP UNTIL INKEY$ <> ""
END
```

PSET ステートメントで 1 つ 1 つのピクセルを描いていくと、どんな図形でも描くことができます。しかしこのような方法では、非常に多数のピクセルで構成される図形では、描画速度がかなり遅くなります。これを避けるために、Visual Basic for MS-DOS では単純な図形 (線、四角形、楕円など) を高速に描画するステートメントが用意されています。

LINE による線の描画

PSET または **PRESET** ステートメントは画面上の 1 点しか扱わないため、1 つの座標 (x, y) しか指定しませんでした。**LINE** ステートメントは、線の両端の 2 つの座標を指定して直線を描きます。**LINE** ステートメントのもっとも単純な形式は、次のようになります。

LINE (*x1!,y1!*) - (*x2!,y2!*)

(*x1!, y1!*) には線の一方の端の座標を指定し、(*x2!, y2!*) にはもう一方の端の座標を指定します。例えば、次のステートメントは座標 (10, 10) から座標 (150, 200) まで直線を描きます。

```
LINE (10, 10)-(150, 200)
```

2 つの座標は、どちらを先に指定してもかまいません。(*x1!, y1!*) から (*x2!, y2!*) までの線は、(*x2!, y2!*) から (*x1!, y1!*) までの線と同じです。上のステートメントは、次のように書くこともできます。

```
LINE (150, 200)-(10, 10)
```

グラフィックスを扱う場合にも、テキスト画面におけるカーソルのように、次にグラフィックスが出力される位置が記憶されています。この位置を、グラフィックスの出力位置と呼びます。座標を指定する順序は、線を描いたあとのグラフィックス出力位置に影響します。

STEP キーワードの指定

ここまでは、画面座標を画面左上隅の座標 (0, 0) を基準とする絶対座標として扱ってきました。次のグラフィックス ステートメントで **STEP** キーワードを指定すると、**STEP** の後ろに続く座標は、そのときのグラフィックス出力位置からの相対座標として扱われます。

CIRCLE	**GET**	**LINE**	**PAINT**
PRESET	**PSET**	**PUT**	

例えば、幅が 1 ピクセルのブラシを使って画面に図形を描くとき、描画を終えたときのブラシの位置、つまり最後に参照した点が、次の“グラフィックスの出力位置”になります。次のステートメントを実行すると、座標 (100, 150) のピクセルにグラフィックスの出力位置が移動します。

```
LINE (10, 10)-(100, 150)
```

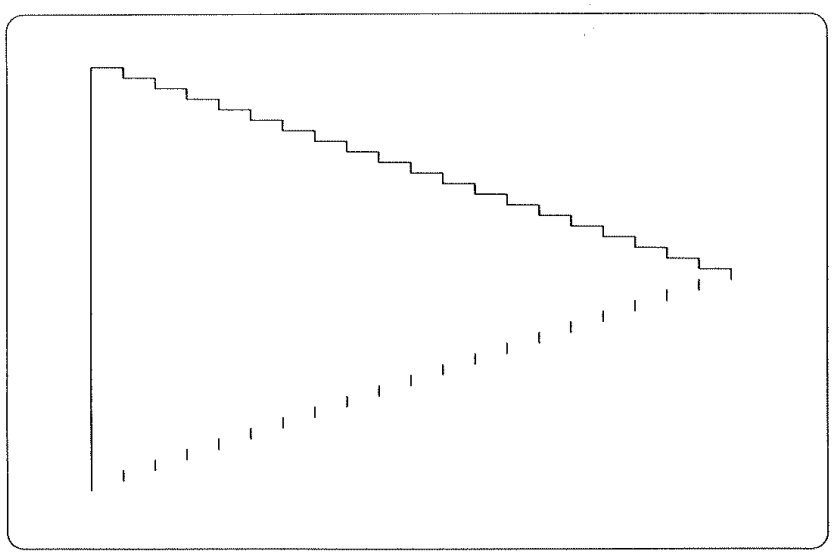
この状態で、次のグラフィックス ステートメントを実行すると、**PSET** ステートメントで描かれる点は、画面の左上の (20, 20) の位置には描かれません。**STEP** キーワードによって、座標 (20, 20) は、最後に参照した点の座標 (100, 150) からの相対座標になります。この点は、絶対座標 (100 + 20, 150 + 20)、つまり (120, 170) に描かれます。

```
PSET STEP(20, 20)
```

この例でわかるように、そのときのグラフィックス出力位置は、直前のグラフィックス ステートメントによって決まります。また次の例のように、同じステートメントで、線を連続して描くこともできます。

```
SCREEN 88

'画面の左下隅から左上隅まで線を引きます.
LINE (10, 399) -STEP(10, 0)

'画面の左上隅から右端まで, "階段" を描きます.
FOR I% = 1 TO 20
    LINE -STEP(30, 0)          '水平線を引きます.
    LINE -STEP(0, 10)          '垂直線を引きます.
NEXT

'右端から左下隅までとびとびに垂直線を引いていきます.
FOR I% = 1 TO 20
    LINE STEP(-30, 0) -STEP(0, 10)
NEXT

SLEEP                          'キーが押されるのを待ちます.
```

画面例



注　意　上のプログラムの最後にある **SLEEP** ステートメントに注意してください。グラフィックスを出力するプログラムをコンパイルして実行するときは、画面の出力を保持するため、プログラムの終わりにこのステートメントを追加してください。このステートメントがないと、プログラムの終了とともに、画面のグラフィックス出力が消去されてしまう場合があります。

破線と点線の描画

LINE ステートメントでは、次に示す構文の *style%* 引数に“ライン スタイル”を指定すると、破線や点線を描くことができます。次のように指定すると、座標 (*x1!*, *y1!*) から座標 (*x2!*, *y2!*) まで、グラフィックスのデフォルト色で破線や点線を描きます。

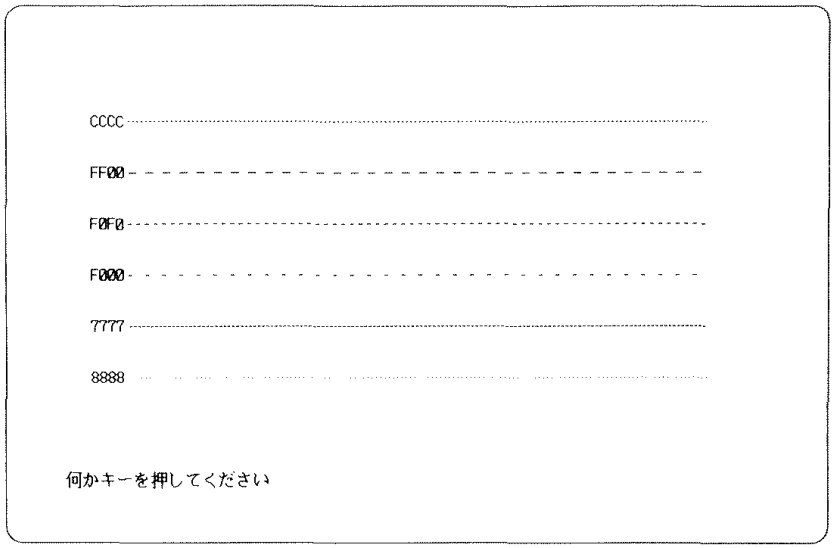
LINE (*x1!*,*y1!*) - (*x2!*,*y2!*), , , *style%*

style% には、10 進数または 16 進数で表した 16 ビットの整数を指定します。**LINE** ステートメントは、引数 *style%* の 2 進数表現によって、実線の部分と空白の部分を表します。ビットの 1 はピクセルを ON にし、ビットの 0 はピクセルを OFF にします。例えば、引数 *style%* に 16 進数の &HCCCC を指定すると、2 進数で 1100110011001100 と表されるため、2 つのピクセルの ON と OFF を交互に繰り返した破線が描かれます。

次の例では、引数 *style%* に異なる値を指定して、いろいろな破線を描きます。

```
'ライン スタイルのデータ
DATA &HCCCC, &HFF00, &HF0F0
DATA &HF000, &H7777, &H8888

Row% = 4
Column% = 4
XLeft% = 60
XRight% = 600
Y% = 56

FOR I% = 1 TO 6
    READ Style%
    LOCATE Row%, Column%
    PRINT HEX$(Style%)
    LINE (XLeft%, Y%)-(XRight%,Y%), , , Style%
    Row% = Row% + 3
    Y% = Y% + 48
NEXT
```

画面例



ボックスの描画

LINE ステートメントを使うと、次のような短いプログラムで、4 本の直線に囲まれたボックスを描くことができます。

```
'1 辺が 120 ピクセルのボックスを描きます.
LINE (50, 50)-(170, 50)
LINE -STEP(0, 120)
LINE -STEP(-120, 0)
LINE -STEP(0, -120)
```

このとき **LINE** ステートメントの **B** (Box) キーワードを使うと、さらに簡単にボックスを描くことができます。次の例では **B** キーワードを使って、上の 4 つの **LINE** ステートメントを使ったときと同じボックスを描きます。

```
'左上隅の座標が (50, 50), 右下隅の座標が (170, 170) のボックスを
'描きます.
LINE (50, 50)-(170, 170), , B
```

LINE ステートメントで **B** キーワードを指定すると、指定した 2 点をつなぐ直線を描くのではなく、2 点を対角 (左上と右下) とする四角形を描きます。

上の例では、**B** キーワードの前にカンマが 2 つ付けられていて、最初のカンマに続く引数 *color&* が省略されたことを示しています。この引数には、線やボックスの 4 辺の色を指定します (色の指定方法については、「色の選択」を参照してください)。**B** キーワードを指定するときも、座標の順序は関係ありません。上の例のボックスは、次のステートメントでも描くことができます。

```
LINE (170, 170)-(50, 50), , B
```

また、**BF** (Box Fill) キーワードを指定すると、ボックスの内側を 4 辺と同じ色で塗りつぶすことができます。

ここで説明したボックスを描くときの構文は、Visual Basic for MS-DOS で四角形の領域を定義する一般的な構文として、**GET** や **VIEW** の各ステートメントでも使われます。

{**GET** | **PUT** | **VIEW**} (*x1!,y1!*) - (*x2!,y2!*), . . .

ここで、(*x1!, y1!*) と (*x2!, y2!*) には、四角形の対角 (左上と右下、または左下と右上) の座標を指定します (**VIEW** ステートメントについては「グラフィックス ビューポートの定義」を、また **GET** と **PUT** の各ステートメントについては「基本的なアニメーション技法」をそれぞれ参照してください)。

2.4 CIRCLE による円と楕円の描画

CIRCLE ステートメントを使うと、円や楕円を描くことができます。また、円弧 (円の一部) や扇形を描くときも、**CIRCLE** ステートメントを使います。これらを応用すれば、いろいろな曲線を描くことができます。

円の描画

円を描くとき、円の中心座標と半径 (円周上の任意の点から中心までの距離) の 2 つの情報が必要です。

CIRCLE ステートメントの基本的な構文は、次のとおりです。

CIRCLE [**STEP**] (*x!,y!*), *radius!*

(*x!, y!*) には円の中心の座標を、*radius!* には円の半径を指定します。次の例では、座標 (200, 100) を中心とする、半径 75 の円を描きます。

```
CIRCLE (200, 100), 75
```

STEP キーワードを使って、次のように指定することもできます。このときの座標は、画面左上隅からの絶対座標ではなく、そのときのグラフィックス出力位置からの相対座標と見なされます。

```
CIRCLE STEP(-120, 0), 75
```

楕円の描画

CIRCLE ステートメントは、画面に描いた円が真円になるように自動的に縦横比を調整しますが、楕円を描くために縦横比を変更することもできます。楕円を描くには、次の構文を使います。

CIRCLE [**STEP**] (*x!,y!*), *radius!*, , , , *aspect!*

radius! と *aspect!* の間のカンマは、色を指定したり、円弧や扇形を描くときに指定する引数が省略されていることを示します (「円弧の描画」, 「色の選択」を参照してください)。

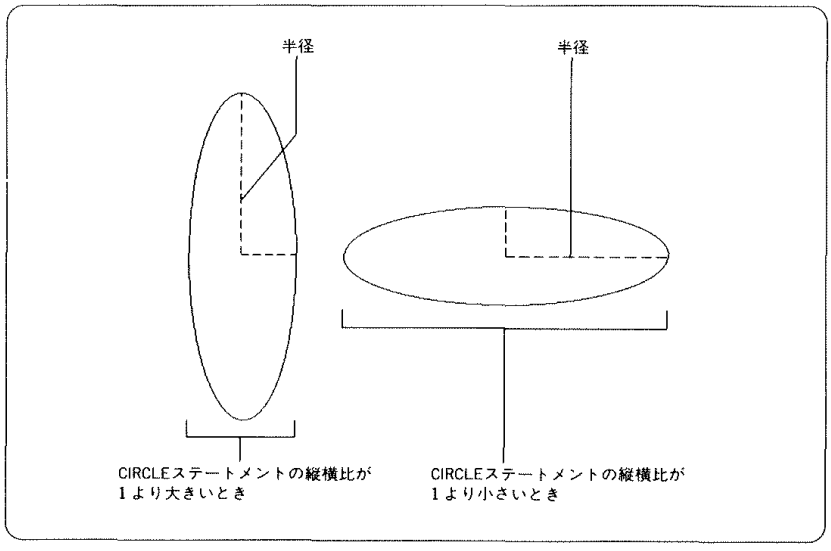
引数 *aspect!* は、円の縦横の大きさの比率を指定します。*aspect!* の値を大きくすると、縦軸の方向に長い楕円を描き、*aspect!* の値を小さくすると、横軸の方向に長い楕円を描きます。楕円は 2 つの半径 (水平方向の x 軸半径と垂直方向の y 軸半径) を持ちますが、**CIRCLE** ステートメントでは、引数 *radius!* の 1 つだけを指定します。*aspect!* が 1 より小さいとき、*radius!* は x 軸半径になり、*aspect!* が 1 以上のとき、*radius!* は y 軸半径になります。

次の例は、*aspect!* の値によって、引数 *radius!* がどのように扱われるかを示します。

```
'画面の左側に楕円を描きます.
CIRCLE (120, 200), 160, , , , 3

'画面の右側に楕円を描きます.
CIRCLE (360, 200), 160, , , , 3/10
```

機種によって、必ずしも真円が描かれない場合があります。このようなときは、*aspect!* の値を変更してください。

画面例



円弧の描画

円弧とは、楕円または円の、円周の一部をいいます。**CIRCLE** ステートメントを使って円弧を描く方法を説明する前に、Visual Basic for MS-DOS で角度がどのように表されるかを知っておく必要があります。

Visual Basic for MS-DOS では、角度の単位としてラジアンを使います。**CIRCLE** ステートメント以外にも、**COS, SIN, TAN** などの三角関数ではラジアンを使います (例外として、**DRAW** ステートメントでは角度を "度" で表します。**DRAW** については、「DRAW: グラフィックス マクロ言語」を参照してください)。

ラジアンは、円の半径と密接な関係があります。“ラジアン (radian)” という言葉も、“半径 (radius)” に由来しています。円周は、2 × π × *radius* で求めることができます。π はおよそ 3.14159265 です。この式から、円周を 1 回転するのに必要な角度 (360 度) をラジアンで表すと、2 × π (約 6.28) となります。度とラジアンの関係は、次のようになります。

度を単位とする角度	ラジアンを単位とする角度
360	2π (約 6.283)
180	π (約 3.142)
90	π/2 (約 1.571)
60	π/3 (約 1.047)

時計の文字盤を考えたとき、**CIRCLE** ステートメントは、図 2.2 に示すように “3 時” の方向から始まって、時計と反対回りに角度を測ります。

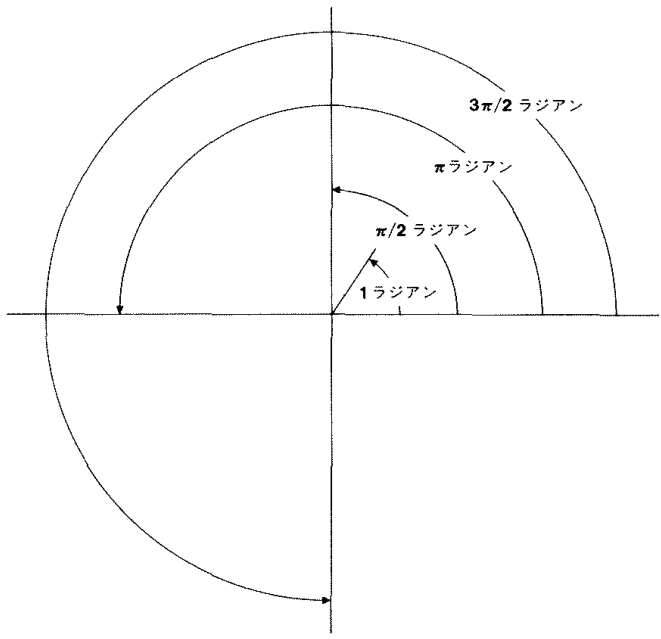


図 2.2　CIRCLE ステートメントにおける角度の表し方

角度の単位を度からラジアンに変換するには、その値に π/180 をかけてください。

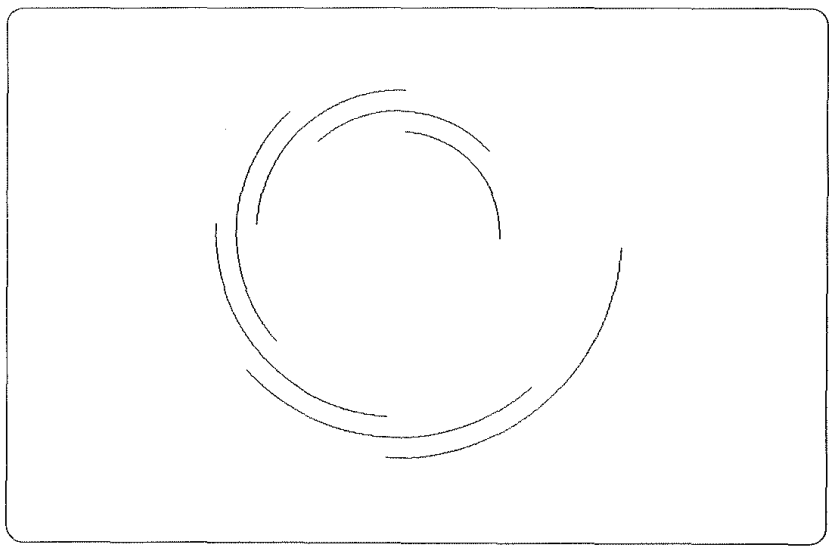
実際に円弧を描くときは、次の構文を使って、円弧の始まり (*start!*) と終わり (*end!*) の角度を指定します。

CIRCLE [**STEP**] (*x!*, *y!*), *radius!*, [*color%*], *start!*, *end!* [, *aspect!*]

次の例は、**CIRCLE** ステートメントを使って 7 つの円弧を描きます。画面例から分かるように、いちばん内側の円弧は“3 時”の方向 (0 ラジアン) から始まり、いちばん外側の円弧は“6 時”の方向 (3π/2 ラジアン) から始まります。

```
CLS

CONST PI = 3.141592653589#        '倍精度の定数

StartAngle! = 0
FOR Radius% = 100 TO 220 STEP 20
    EndAngle! = StartAngle! + (PI / 2.01)
    CIRCLE (320, 100), Radius%, , StartAngle!, EndAngle!
    StartAngle! = StartAngle! + (PI / 4)
NEXT Radius%
```

画面例



扇形の描画

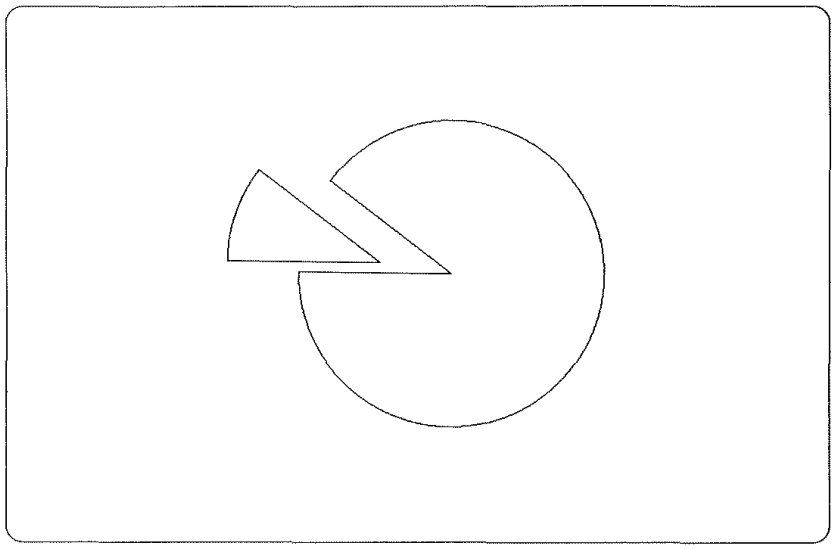
CIRCLE の *start!* または *end!* の各引数に負の値を指定すると、円弧の開始点または終了点と円の中心を線で結ぶことができます。両方の引数に負の値を指定すると、扇形を描くことができます。

次の例は、円から扇形を切り出した図形を描きます。

```
CONST RADIUS = 150, PI = 3.141592653589#

StartAngle! = 2.5
EndAngle! = PI

'扇形を描きます.
CIRCLE (280, 150), RADIUS, , -StartAngle!, -EndAngle!
```

```
'円弧の開始点と終了点の値を交換します.
SWAP StartAngle!, EndAngle!

'円の中心を下に 10 ピクセル, 右に 70 ピクセル移動し, 扇形を切り出した後の
'円を描きます.
CIRCLE STEP(70, 10), RADIUS, , -StartAngle!, -EndAngle!
```

画面例



2.5 DRAW: グラフィックス マクロ言語

Visual Basic for MS-DOS には、これまで説明してきたものとは構造の異なるグラフィックス ステートメントが用意されています。“グラフィックス マクロ言語”である **DRAW** ステートメントです。**DRAW** ステートメントは、グラフィックス ステートメントの 1 つというよりは、それ自体が小型の言語になっています。**DRAW** ステートメントで、1 文字または 2 文字から成るコマンド(“マクロ文字列”)を指定すると、それだけで画面に図形を描き、それに色を塗ることができます。

DRAW ステートメントには、これまでに説明した他のグラフィックス ステートメントと比べて、次のような利点があります。

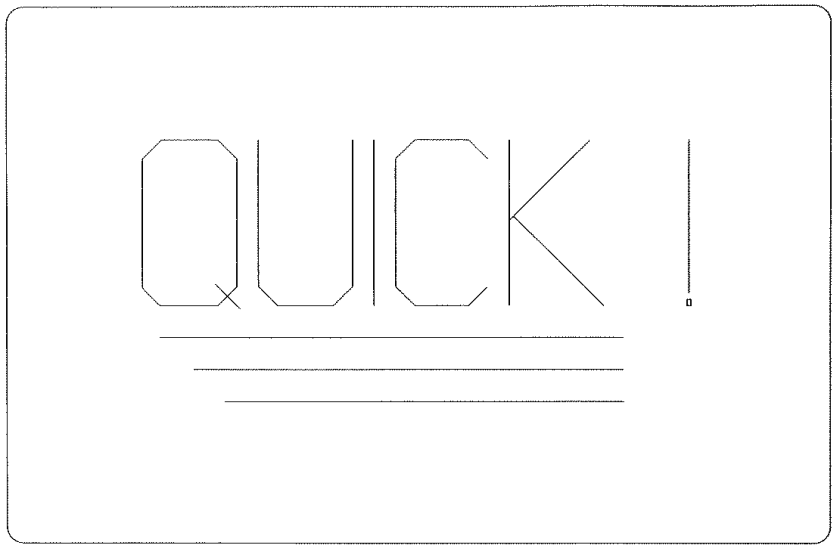
- **DRAW** の引数であるマクロ文字列は、非常にコンパクトです。1 つの短い文字列で、複数の **LINE** ステートメントと同じ出力を得ることができます。
- **DRAW** で作った図形は、**S** マクロ コマンドを使って、容易に縮小/拡大することができます。
- **DRAW** で作った図形は、**TA** マクロ コマンドを使って、何回でも回転させることができます。

マクロ文字列について詳しくは、『リファレンス』またはオンライン ヘルプを参照してください。

次に示すプログラムは、移動マクロ **U, D, L, R, E, F, G, H**, 描画マクロ **B**, カラー マクロ **C** の機能を簡単に紹介します。このプログラムは、テンキーの [1] から [9] を使って、水平線，垂直線，対角線をそれぞれ異なる色で描きます。

```
SCREEN 88
'テンキー上のキーと [SPACE] キーの値を定義します.
CONST UP = &h38, DOWN = &h32, LFT = &h34, RGHT = &h36
CONST UPLFT = &h37, UPRGHT = &h39, DOWNLFT = &h31
CONST DOWNRGHT = &h33, SPACEBAR = " "

'Plot$ = "" のときは，線を引きます. Plot$ = "B" のときは,
'グラフィックス カーソルを移動しますが，線は引きません.
Plot$ = ""
PRINT "テンキーを使って線を引きます."
PRINT "[SPACE] キーで描画の ON/OFF を切り換えます."
PRINT "[ENTER] で始めます. [q] を押すと，プログラムを終了します."
DO : LOOP WHILE INKEY$ = ""

CLS

DO
    SELECT CASE KeyVal$
        CASE Null$ + CHR$(UP)
            DRAW Plot$ + "C2 U2"
        CASE Null$ + CHR$(DOWN)
            DRAW Plot$ + "C2 D2"
        CASE Null$ + CHR$(LFT)
            DRAW Plot$ + "C4 L2"
        CASE Null$ + CHR$(RGHT)
            DRAW Plot$ + "C4 R2"
        CASE Null$ + CHR$(UPLFT)
            DRAW Plot$ + "C6 H2"
        CASE Null$ + CHR$(UPRGHT)
            DRAW Plot$ + "C6 E2"
        CASE Null$ + CHR$(DOWNLFT)
            DRAW Plot$ + "C6 G2"
        CASE Null$ + CHR$(DOWNRGHT)
            DRAW Plot$ + "C6 F2"
        CASE SPACEBAR
            IF Plot$ = "" THEN Plot$ = "B" ELSE Plot$ = ""
        CASE ELSE
            'テンキー, [SPACE] キー, "[q]" 以外のキーが押されたときは,
            '何も行いません.
    END SELECT

    KeyVal$ = INKEY$

LOOP UNTIL KeyVal$ = "q"

END
```

画面例



2.6 グラフィックス ビューポートの定義

これまでにあげたグラフィックス プログラムの例は、画面の左上隅を基準とする絶対座標を使って、表示画面全体に描画していました。

VIEW ステートメントを使うと、物理画面の中に“グラフィックス ビューポート”と呼ぶ小さな画面を定義することができます。グラフィックス ビューポートを定義すると、すべてのグラフィックス操作がこのビューポート内で行われるようになります。ビューポートの境界の外にあるグラフィックス出力は、すべて切り取られます。つまり、ビューポートの外に描かれた点は、すべて無視されます。ビューポートの利点として、次の 2 つがあげられます。

- ビューポートを使うと、グラフィックスを表示する画面のサイズや位置を自由に変えることができます。
- ステートメント CLS 1 を使って、ビューポートの外の画面に影響を与えずに、ビューポート内の画面を消去することができます。

注 意 テキストの表示領域を定義する“テキスト ビューポート”については、『プログラミング ガイド』第 11 章「デスクトップ出力とキーボード入力」を参照してください。

VIEW ステートメントの構文を次に示します。**STEP** キーワードが使えないことに注意してください。

VIEW [[**SCREEN**] (*x1!*, *y1!*) - (*x2!*, *y2!*) [, [*color&*] , [*border&*]]]

座標 (*x1*, *y1*) と (*x2*, *y2*) には、Visual Basic for MS-DOS で四角形の領域を定義する標準的な構文を使って、ビューポートの対角の座標を指定します (詳しくは「ボックスの描画」を参照してください)。また、省略可能な引数である *color&* と *border&* を指定すると、ビューポート内の色と境界線の色を指定することができます。色の設定や変更については、「色の選択」を参照してください。

VIEW ステートメントのすべての引数を省略すると、画面全体がビューポートになります。また、**SCREEN** キーワードを省略すると、すべてのピクセルの座標がビューポートに対する相対座標になります。

例えば、次の **VIEW** ステートメントを実行したとき、それに続く **PSET** ステートメントでセットされたピクセルは、ビューポートの左上隅から下に 10 ピクセル、右に 10 ピクセルの位置に表示されます。

```
VIEW (50, 60)-(150, 175)
  .
  .
  .
PSET (10, 10)
```

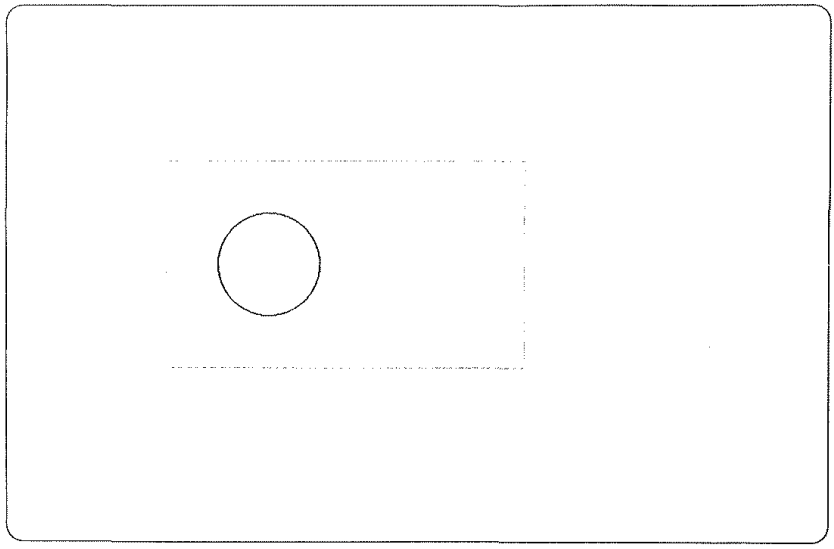
このピクセルの絶対座標は、(50 + 10, 60 + 10) つまり (60, 70) になります。ただし、**VIEW** ステートメントで **SCREEN** キーワードを指定すると、すべての座標が絶対座標 (画面の左上隅からの距離) になります。この場合、**PSET** ステートメントで指定したピクセルはビューポートの外側 (画面の左上隅から下に 10 ピクセル，右に 10 ピクセル) になり、表示されません。

```
VIEW SCREEN (50, 60)-(150, 175)
PSET (10, 10)
```

次の 2 つのプログラムの出力を比較すると、**VIEW** と **VIEW SCREEN** の違いがよくわかります。

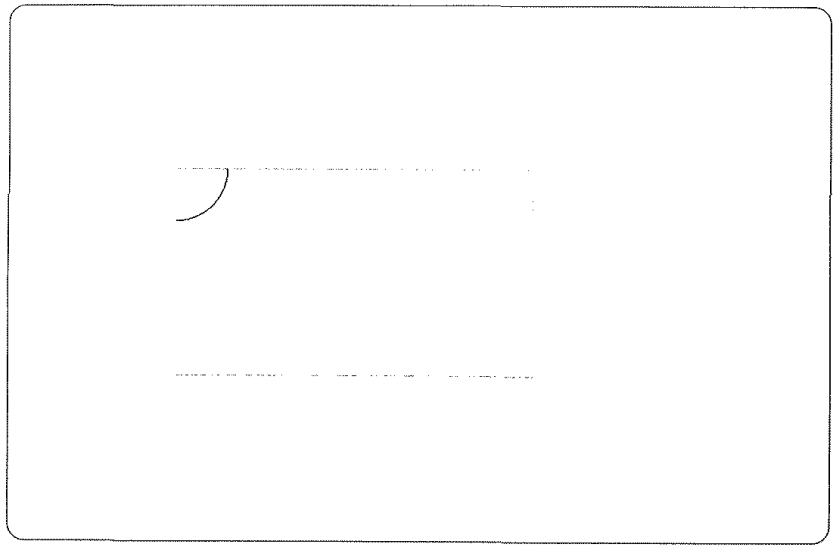
```
'この円の中心の絶対座標は (100 + 100, 100 + 100), すなわち (200, 200)
'になります.
VIEW (100, 100)-(450, 300), , 1
CIRCLE (100, 100), 50
```

画面例



```
'この円の中心の絶対座標は (100, 100) になるため，ビューポートには円の一部
'しか表示されません．
VIEW SCREEN (100, 100)-(450, 300), , 1
CIRCLE (100, 100), 50
```

画面例



ビューポートの外に出力されたグラフィックスは、ビューポートの境界で切り取られて、画面には表示されないことに注意してください。

2.7 ビューポート座標の再定義

ここでは、**WINDOW** ステートメントを使って、座標系を再定義する方法を説明します。

ここまでの説明では、画面上のピクセルの座標を、画面の左上隅 (**VIEW** ステートメントを実行したときは、現在のビューポートの左上隅) からの物理的な距離で表しました。これらの座標を“物理座標”と呼びます。物理座標の基準となる“原点”は、画面またはビューポートの左上隅にあり、(0, 0) という座標を持ちます。

この座標系では、画面の右に行くほど x (水平座標) の値が大きくなり、下に行くほど y (垂直座標) の値が大きくなります。画面上では、一般にこのような座標系が使われますが、グラフを描くときなどは不便な場合があります。例えば、数学で使うデカルト座標 (直交座標) では、y の値がグラフの上に行くほど大きくなり、下に行くほど小さくなります。

WINDOW ステートメントを使うと、座標系を再定義することができます。**WINDOW** ステートメントの構文を次に示します。

WINDOW [[**SCREEN**] (*x1!*,*y1!*) - (*x2!*,*y2!*)]

y1!, *y2!*, *x1!*, *x2!* には、ビューポートの上端，下端，左端，右端の新しい座標を示す実数を指定します。これらの数値を“ビューポート座標”と呼びます。例えば、ビューポートが画面全体に設定されている状態で次のステートメントを実行すると、画面の上端と下端の境界は、それぞれ y = -15 と y = 10 になり、左端と右端の境界は、それぞれ x = -25 と x = 5 になります。

```
WINDOW (-25, -15)-(5, 10)
```

この **WINDOW** ステートメントを使うと、y の値は画面の上に行くほど大きくなります。ただし **SCREEN** キーワードを指定すると、y の値は画面の下に行くほど大きくなります。次の図は、**WINDOW** ステートメントの **SCREEN** キーワードを指定したときと指定しないときで、同じ座標を指定して描いた直線がどのように描画されるかを示しています。このとき、2 つのステートメントによって、画面の隅の座標が変わることに注意してください。引数を指定せずに **WINDOW** ステートメントを実行すると、通常の物理座標系に戻ります。

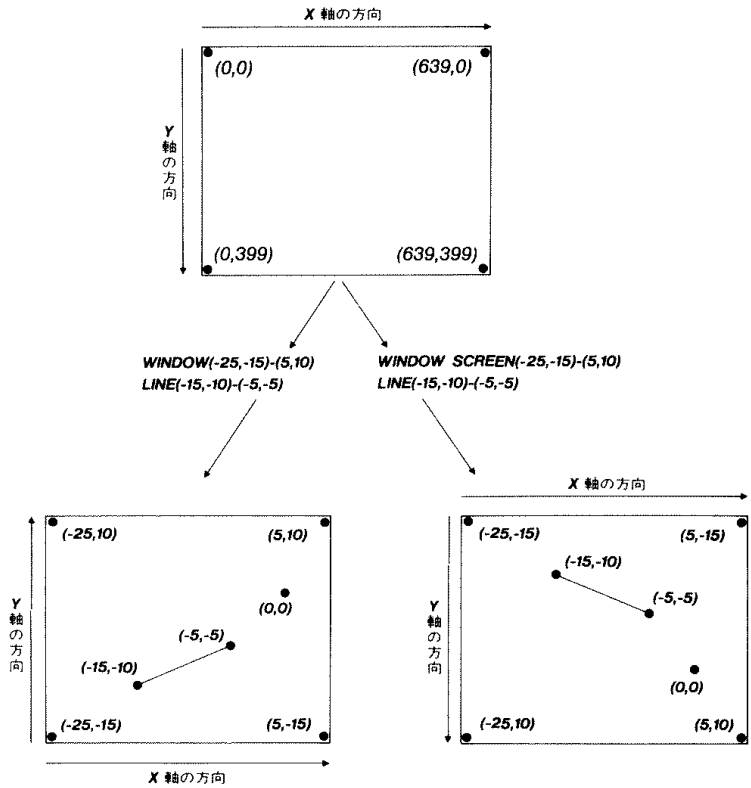


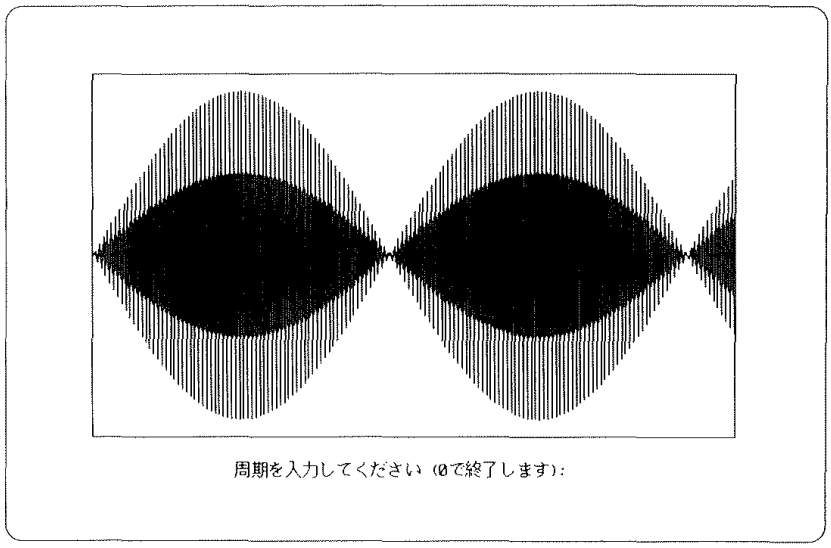
図 2.3　WINDOW と WINDOW SCREEN の違い

次の例では、0 ラジアンからπラジアン (0° から 180°) までのサイン カーブを描き、**VIEW** と **WINDOW** の 2 つのステートメントを使って、プログラムを単純化しています。

```
CLS

'グラフを描くためのビューポートを定義します.
VIEW (20, 4)-(620, 344), , 1
CONST PI = 3.141592653589#

'0 ラジアンからπラジアンまでのサインカーブを描くために十分なウィンドウを
'定義します.
WINDOW (0, -1.1)-(PI, 1.1)
Style% = &HFF00                 '破線を描くために使います.
VIEW PRINT 23 TO 24             '23 行と 24 行のテキスト出力をスクロール
                                'させます.
DO
    PRINT TAB(20);
    INPUT "周期を入力してください (0で終了します):", Cycles
    CLS
    LINE (PI, 0)-(0, 0), , , Style%     'x 軸を描きます.
```

```
    IF Cycles > 0 THEN
        '(0, 0) から始めてグラフを描きます.
        FOR X=0 TO PI STEP .01
            Y = SIN (Cycles * X)          'y 座標を計算します.
            LINE -(X, Y)                  '新しい点まで線を引きます.
        NEXT X
    END IF
LOOP WHILE Cycles > 0
```

画面例



座標値の順序

四角形の領域を定義する他のグラフィックス ステートメント (**GET, LINE, VIEW**) と同様、**WINDOW** ステートメントで指定する座標の順序は、それほど重要ではありません。

次の 2 つの例は、どちらも同じ結果になります。

```
VIEW (100, 20)-(300, 120)
WINDOW (-4, -3)-(0, 0)

VIEW (300, 120)-(100, 20)
WINDOW (0, 0)-(-4, -3)
```

ビューポート座標と物理座標の獲得

PMAP と **POINT** の 2 つの関数は、物理座標とビューポート座標を求めるときに役立ちます。**POINT**(*number%*) は、*number%* 引数に指定した値に従って、現在のグラフィックス出力位置の物理座標またはビューポート座標のいずれかの値を返します。**PMAP** 関数は、物理座標のビューポート座標への変換と、ビューポート座標の物理座標への変換を行います。**PMAP** 関数の返す物理座標の値は、常に現在のビューポートに対する相対座標になります。

次の例は、**POINT** (*number%*) の引数 *number%* に、0, 1, 2, 3 の値を指定したときに返される値の違いを示します。

```
'ビューポート座標を定義します.
WINDOW (-10, -30)-(-5, -10)
'ビューポート座標 (-9,-28) から (-6, -24) まで線を引きます.
LINE (-9, -28)-(-6, -24)

PRINT "グラフィックス出力位置の物理座標の x 座標 = " POINT(0)
PRINT "グラフィックス出力位置の物理座標の y 座標 = " POINT(1)
PRINT
PRINT "グラフィックス出力位置のビューポート座標の x 座標  = " POINT(2)
PRINT "グラフィックス出力位置のビューポート座標の y 座標  = " POINT(3)

END
```

出力例

```
グラフィックス出力位置の物理座標の x 座標 = 511
グラフィックス出力位置の物理座標の y 座標 = 279
グラフィックス出力位置のビューポート座標の x 座標 = -6
グラフィックス出力位置のビューポート座標の y 座標 = -24
```

上の例と同じビューポートが定義されているとき、**PMAP** 関数が返す値を次に示します。

```
'ビューポート座標の x 座標 -6 を物理座標に変換して表示します.
PhysX% = PMAP(-6, 0)
PRINT PhysX%

'ビューポート座標の y 座標 -24 を物理座標に変換して表示します.
PhysY% = PMAP(-24, 1)
PRINT PhysY%

'上で求めた x 座標をビューポート座標に再変換して表示します.
ViewX% = PMAP(PhysX%, 2)
PRINT ViewX%

'上で求めた y 座標をビューポート座標に再変換して表示します.
ViewY% = PMAP(PhysY%, 3)
PRINT ViewY%
```

出力例

```
511
279
-6
-24
```

2.8 色の選択

色を示す値には、色番号，色値，パレット番号の3種類があります。色番号と色値はある特定の色を示す値であるのに対し、パレット番号は論理的な値で、あるパレット番号が実際にどんな色を示すかは、そのときの設定によって決まります。色番号は、テキスト画面の文字の色や背景色と、フォーム上でのオブジェクトの色の指定に使う0〜15の整数で、グラフィックスでは使いません。

パレットとパレット番号

実際に画面に表示される色は、R(赤)/G(緑)/B(青)の組み合せで決まります。RGBの物理的な組み合せを示す値を、“色値”と呼びます。しかし、図形を描くたびに色値を指定するのはめんどうです。そこでグラフィックスでは、“パレット”と呼ばれるテーブルを使って色を設定します。パレットは、絵具のパレットにたとえることができます。1枚のパレットには、いくつかの色の絵具があります。これらの色には、“パレット番号”と呼ばれる番号が付けられていて、それぞれのパレット番号には、実際に表示される色を示す値である色値が割り当てられています。画面に描画を行うステートメントでは、このパレット番号で色を指定します。パレット番号で色を指定するステートメントには、次のようなものがあります。

CIRCLE	**COLOR**	**DRAW**	**LINE**
PAINT	**PRESET**	**PSET**	**VIEW**

パレットのデフォルトの設定については、第1章「PC-9801シリーズ固有の仕様」を参照してください。

パレットの設定の変更

パレットの設定を変更するには、つまりパレット番号と色値の割り当てを変更するには、**PALETTE** ステートメントまたは **PALETTE USING** ステートメントを使います。**PALETTE** ステートメントは1つのパレット番号に対する色の設定を変更し、**PALETTE USING** ステートメントは一度にパレット全体を変更します。**PALETTE** ステートメントの構文は、次のとおりです。

PALETTE [*color%, color-value%*]

color% にはパレット番号、*color-value%* にはその番号に割り当てる色値を指定します。あるパレット番号の色を変更すると、そのとき画面に表示されている図形のうち、そのパレット番号で描かれたものの色もすべて変更されます。言い換えると、先に画面に図形を表示してから、パレットを変更して一度に色を変えることができます。次に例を示します。

```
SCREEN 1, 3                          '4096 色中 16 色表示
LINE (50, 50)-(150, 150), 4          '赤い線を描きます.
SLEEP 1                              'プログラムの実行を休止します.
PALETTE 4, &HF0                      'パレット番号 4 に, 緑を示す色値を
                                     '割り当てます.
```

PALETTE USING ステートメントを使うと、すべてのパレット番号に割り当てられた色を、一度に変更できます。

次に示す例は、**PALETTE USING** ステートメントを使って、パレット番号 1 ～ 15 に割り当てられた色を順に変更することによって、画面の色を変えています。はじめに 15 個の同心円を画面に表示し、次に各パレット番号の色を実行のたびに 1 ずつ変化させています。

```
DIM PA&(15)
SCREEN 1, 3                      '4096 色中 16 色表示
'デフォルトのパレットの色値
DATA &HFFF, &HFF0, &HF0F, &HF00, &H0FF, &H0F0, &H00F, &H777
DATA &HCCC, &HAA0, &HA0A, &HA00, &H0AA, &H0A0, &H00A, &H000
FOR CV% = 15 TO 1 STEP -1            '同心円を描きます.
    READ PA&(CV%)
    CIRCLE (320, 200), 10 * CV%, CV%
    PAINT (320, 200), CV%
NEXT

J% = 15
WHILE INKEY$ = ""                    'キーが何も押されない限り,
    FOR I% = 1 TO 15                 'カラー パレットのシフトを続けます.
        J% = J% + 1
        READ PA&(I%)
        IF J% = 16 THEN
            J% = 0
            RESTORE
        END IF
        PALETTE USING PA&
    NEXT I%
WEND
```

2.9 図形の塗りつぶし

「ボックスの描画」では **LINE** ステートメントの **BF** キーワードを使って、中を塗りつぶしたボックスを描く方法を説明しました。

```
'四角形を描き，その内部をパレット番号 1 の色で塗りつぶします．
LINE (50, 50)-(110, 100), 1, BF
```

PAINT ステートメントを使うと、線で囲まれた図形の内部を任意の色で塗ることができます。また、任意の模様 (縞模様や格子模様) を指定して図形の内部を塗りつぶすこともできます。

色による塗りつぶし

線で囲まれた任意の図形の内部を色で塗りつぶすには、次の構文を使います。

PAINT [STEP](*x!,y!*) [, [*paint%*], [*bordercolor%*]]

(*x!, y!*) には、塗りつぶしたい図形の内部にある点の座標を指定します。これは対象とする図形の内側の点であれば、どの点でもかまいません。*paint%* には、塗りつぶす色を示すパレット番号を指定します。*bordercolor%* には、図形の境界線の色を示すパレット番号を指定します。

次の例は、円を描き、その内部を塗りつぶします。

```
CIRCLE (160, 100), 50, 1
PAINT (160, 100), 2, 1
```

PAINT ステートメントには、次の 3 つの規則があります。

- **PAINT** ステートメントには、図形の内側の点の座標を指定します。例えば、次の 3 つのステートメントは、すべて円の内側の点を指定しているため、上の例の **PAINT** ステートメントと同じ効果を持つことになります。

  ```
  PAINT (150, 90), 2, 1
  PAINT (170, 110), 2, 1
  PAINT (180, 80), 2, 1
  ```

  これに対して、次のステートメントでは、座標 (5, 5) が円の外側の点であるため、円の内側を除く画面全体を塗りつぶします。円の内側だけが現在の背景色のまま残されます。

  ```
  PAINT (5, 5), 2, 1
  ```

  PAINT ステートメントで指定した座標が図形の境界線上の 1 点を表す場合、色による塗りつぶしは行われません。

  ```
  LINE (50, 50)-(150, 150), , B        '四角形を描きます．
  PAINT (50, 100)              '座標 (50, 100) は四角形の辺 (輪郭) 上に
                               'あるため，この四角形の内部は塗りつぶされま
                               'せん．
  ```

- 塗りつぶされる図形は、完全に閉じていなければなりません。図形が閉じていないと、塗った色が“染み出す”ことになり、画面やビューポート全体(または最初に指定した図形を含む、より大きな閉じた図形)全体に拡がっていきます。

  次のプログラムは、**CIRCLE** ステートメントでわずかに(右側に)すきまのある楕円を描き、次に **LINE** ステートメントを使って、この不完全な楕円をボックスで囲みます。色による塗りつぶしは楕円の内側から始まりますが、この色は楕円のすき間から染み出し、ボックス全体に拡がっていきます。

```
CONST PI = 3.141592653589#
CIRCLE (300, 100), 80, , 0, 1.9 * PI, 3
LINE (200, 10)-(400, 190), , B
PAINT (300, 100)
```

- ある図形の内部を、その輪郭と違う色で塗りつぶす場合は、*bordercolor&* 引数を指定して、その境界を指定する必要があります。

  例えば、次のプログラムはパレット番号 1 の色で輪郭を描いた三角形の内部を、パレット番号 2 の色で塗りつぶします。しかし、**PAINT** ステートメントに色を塗りつぶす境界の指定がないため、画面全体が塗りつぶされてしまいます。

```
LINE (10, 25)-(310, 25), 1
LINE -(160, 175), 1
LINE -(10, 25), 1

PAINT (160, 100), 2
```

  ここで **PAINT** ステートメントを次のように変更すると、図形の境界線を残して、三角形の内部だけを塗りつぶすことができます。

```
PAINT (160, 100), 2, 1
```

  PAINT ステートメントで境界線の色を指定する必要がないのは、図形の内部の色と境界線の色が同じであるときだけです。

```
LINE (10, 25)-(310, 25), 1
LINE -(160, 175), 1
LINE -(10, 25), 1

PAINT (160, 100), 1
```

塗りつぶし模様の指定: タイリング

PAINT ステートメントでは、塗りつぶしの色だけでなく、模様を指定することもできます。この模様付けのことを“タイリング”と呼びます。これは、床にタイルを貼りながら模様を作って行く作業に似ているからです。タイリングを行う場合、**PAINT** の引数 *paint* に、パレット番号ではなく、文字列式を指定します。*paint* には任意の文字列式を指定できますが、通常は次に示す形式でタイルの模様を定義します。

CHR$**(*code1%*) + **CHR$(*code2%*) + **CHR$**(*code3%*) +...+ **CHR$**(*coden%*)

ここで、*code1%, code2%,...* は 8 ビットの整数です。これらの 8 ビット整数の求め方については、この後で説明します。

スクリーン モードと色数

模様の単位となるタイルは、横 8 ピクセル分ずつを、文字 (8 ビット) を使って指定します。この場合、そのときのスクリーン モードで同時表示可能な色数によって、何文字で横 8 ピクセル分が表せるかが決まります。例えば、1 ピクセルの色を青 (B)，緑 (G)，赤 (R)，輝度 (I) の 4 枚のグラフィックス プレーンで表す構成の場合には、4 文字分 (32 ビット) で横 8 ピクセル分が表せることになります。

タイルの定義

タイルは、模様の単位となるものです。“指定された範囲に模様を付ける”ということは、その範囲をタイルで埋めていくということです。

タイルを使うには、どのようなタイルを使うのかを指定しなければなりません。これを“タイルの内容の定義”と呼びます。1 つのタイルは、ピクセルによって構成される四角の格子です。そこで、タイルの定義は、この格子内の各ピクセルが光る (ON) / 光らない (OFF)、またどんな色で光るかを指定することになります。

次に、タイルを定義する方法を、模様の定義と色の定義方法に分けて説明します。

タイルの模様の定義

ここでは、タイル全体の模様の定義を説明し、色のことは考えません。実際にタイルを使うには色の指定が必要ですが、色の指定は別に説明します。

タイルの定義は、行単位で行います。1 行には横に 8 ピクセルと決まっています。一方、縦には 64 行まで使うことができます。つまり、縦の大きさは 1～64 ピクセルの範囲で指定することができます。

文字“M”の形をした模様を作ることを考えてみましょう。つまり、ピクセルの ON/OFF を組み合わせて、“M”の形にします。横に 8 ピクセル使うことを考えると次の図のようになります。ボックス内の四角 (■) は、そのピクセルが光っている (ON) ことを示します。

■					■		
■	■			■	■		
■		■	■		■		
■					■		
■					■		

行数は、最大 64 行までの範囲で自由に指定できます。ここでは、タイルをならべるときの文字“M”の間のすきまを考えて、下に 1 行余分にとることにします。つまり、横に 8、縦に 6 ピクセル使います。

この模様をタイル パターンの定義とするには、ピクセルの ON/OFF を行単位で 8 ビットの数値に変換します。上の図で、ピクセルの ON を 1、OFF を 0 に対応させると、次の図のようになります。

1	0	0	0	0	1	0	0
1	1	0	0	1	1	0	0
1	0	1	1	0	1	0	0
1	0	0	0	0	1	0	0
1	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0

それぞれの行を、8 桁の 2 進数 (8ビット) と見なします。8 桁の 2 進数は、4 桁ごとにまとめて 2 桁の 16 進数とすると、簡単に表現することができます。

```
10000100 = &H84
11001100 = &HCC
10110100 = &HB4
10000100 = &H84
10000100 = &H84
00000000 = &H00
```

タイルの模様は、この数値を ASCII 文字に変換することによって定義します。数値から ASCII 文字への変換には、**CHR$** 関数を使います。

```
Tile$ = CHR$(&H84) + CHR$(&HCC) + CHR$(&HB4) + CHR$(&H84)
  ⇒ + CHR$(&H84) + CHR$(&H00)
```

このステートメントは、8 ピクセルごとにひとまとまりにした 6 つの文字で、8×6 ドットの格子を定義します。考え方の上では、この文字列がタイルの模様となり、**PAINT** ステートメントに引数として指定することになります。

```
PAINT (X, Y), Tile$
```

ただし、この指定はこのままでは正しくありません。それぞれのピクセルの色を指定していないからです。

タイルの色の定義

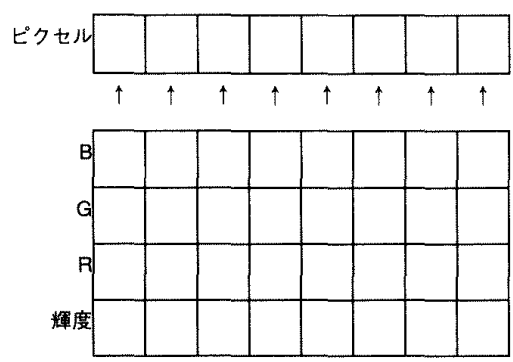
タイルを定義するには、模様だけでなく色も指定する必要があります。パレットの設定がデフォルトで、同時に最大 16 色表示できる場合、1 つのピクセルは、次のように考えることができます。

- 1 つのピクセルは、RGB (赤, 緑, 青) の 3 原色から成ります。それぞれの色の ON / OFF を指定することができるので、結果として 8 色を表現することができます。
- 1 つのピクセルには、輝度 (明るい / 暗い) を指定することができます。

1つのピクセルには、R(赤), G(緑), B(青), 輝度の4つの要素のON/OFFを指定できるので、1ピクセルあたり4ビットの指定が必要になります。その結果、16種類の色を表現することができます。

輝度	R	G	B	色
0	0	0	0	黒
0	0	0	1	青
0	0	1	0	緑
0	0	1	1	水
0	1	0	0	赤
0	1	0	1	紫
0	1	1	0	黄
0	1	1	1	白
1	0	0	0	灰
1	0	0	1	青(明)
1	0	1	0	緑(明)
1	0	1	1	水(明)
1	1	0	0	赤(明)
1	1	0	1	紫(明)
1	1	1	0	黄(明)
1	1	1	1	白(明)

このように考えると、タイルの1行は、8ビット数が4つ重なった層によって構成されることになります。つまり、1行(8ピクセル)が、4層の“ビット プレーン”によって表現されます。



そこで、タイルの1行(8ピクセル)は、前に説明したような1組の8ビット数ではなく、4組の8ビット数によって指定されることになります。つまり、4バイトを1組とした文字列で定義されます。次に、黄色と紫のモザイク模様を作る手順を示します。

1. 模様タイルのピクセル行を定義します。各ピクセルが4ビットを必要とし、各ビットは、異なるプレーンに置かれます。黄色はR(赤)+G(緑)で低輝度にし、紫はR(赤)+B(青)で高輝度にします。

ピクセル	紫	紫	黄	黄	黄	黄	紫	紫
	↑	↑	↑	↑	↑	↑	↑	↑
B	1	1	0	0	0	0	1	1
G	0	0	1	1	1	1	0	0
R	1	1	1	1	1	1	1	1
輝度	0	0	1	1	1	1	0	0

この結果、4つの8ビット数が得られます。

```
11000011 = &HC3
00111100 = &H3C
11111111 = &HFF
00111100 = &H3C
```

これらの数値を使って、タイルの1行を構成する文字列を作ります。

```
Row1$ = CHR$(&HC3) + CHR$(&H3C) + CHR$(&HFF) + CHR$(&H3C)
```

2. 模様を作るために、黄色と紫が逆の行を作ります。

ピクセル	黄	黄	紫	紫	紫	紫	黄	黄
	↑	↑	↑	↑	↑	↑	↑	↑
B	0	0	1	1	1	1	0	0
G	1	1	0	0	0	0	1	1
R	1	1	1	1	1	1	1	1
輝度	1	1	0	0	0	0	1	1

4つの8ビット数が得られます。

```
00111100 = &H3C
11000011 = &HC3
11111111 = &HFF
11000011 = &HC3
```

これらの数値を使って、タイルの1行を構成する文字列を作ります。

```
Row2$ = CHR$(&H3C) + CHR$(&HC3) + CHR$(&HFF) + CHR$(&HC3)
```

3. 手順1,2で作った2つの4バイト(8ビット×4)の文字列(Row1$とRow2$)を必要な行数だけ繰り返すことによってタイルを定義します。

```
Tile$ = Row1$ + Row1$ + Row2$ + Row2$
```

紫	紫	黄	黄	黄	黄	紫	紫
紫	紫	黄	黄	黄	黄	紫	紫
黄	黄	紫	紫	紫	紫	黄	黄
黄	黄	紫	紫	紫	紫	黄	黄

これは、4行のタイルで、4つの4バイト文字列によって定義されています。

次の例は、はじめにボックスを描き、次にその内部に上の手順で作った模様を付けます。

```
DIM Row$(1 TO 4)
SCREEN 1

'紫と黄の繰り返しによる 2 つのピクセル行.
Row$(1) = CHR$(&HC3) + CHR$(&H3C) + CHR$(&HFF) + CHR$(&H3C)
Row$(2) = Row$(1)

'模様を反転します.
Row$(3) = CHR$(&H3C) + CHR$(&HC3) + CHR$(&HFF) + CHR$(&HC3)
Row$(4) = Row$(3)

'上で定義したピクセル行から, 模様を作成します.
FOR I% = 1 TO 4
    Tile$ = Tile$ + Row$(I%)
NEXT I%

'ボックスを描き, その内部を定義した模様を付けます.
LINE (50, 50)-(570, 150), , B
PAINT (320, 100), Tile$
```

注　意　色の指定は、パレットの影響を受けます。つまり、デフォルトではパレット番号1は青ですが、これを黄色に変更すると、赤緑青ではなく赤緑黄の組み合せで表示色が決まります。

2.10 基本的なアニメーション技法

画面上に描いた図形を消して新しい位置に再び表示し、その操作を何度も繰り返すと、まるでその図形が画面上を移動しているように見えます。これを“アニメーション”(動画) と呼びます。

色を使い分けることによって、簡単なアニメーションを作ることができます。例えば、まず **CIRCLE** ステートメントを使って円を描き、背景色で再び同じ円を描くと、その円は画面から消えたように見えます。次に、中心を別の位置に移動してもう一度円を描くと、画面上でその円が動いたように見えます。

このテクニックは単純な図形の場合には便利ですが、複雑なイメージをアニメーションにするときには不十分です。個別のグラフィックス ステートメントの処理は速くても、数多くのものを組み合わせると、全体として処理が遅くなります。単純な円やボックスだけを速く動かすことはできても、複数の形や模様を持つ図形の移動はそれだけ遅くなります。さらに、背景のあるところに図形を描くとき、描いた図形を消したあとで背景を元に戻すのは、かなりむずかしくなります。

アニメーションを高速化するための手段として、**GET** と **PUT** の 2 つのステートメントが用意されています。これらのステートメントを使うには、まず、**PSET, LINE, CIRCLE, PAINT** などのステートメントで画面上にイメージを作成し、**GET** ステートメントでそのイメージをメモリにコピーします。その後で **PUT** ステートメントを使い、画面またはビューポート上の任意の位置に、メモリに保存されたイメージを再表示します。

イメージの保存: GET

イメージを保存するには、元になるイメージを画面上に作成し、そのイメージ全体を囲む四角形の領域を、**GET** ステートメントを使ってメモリにコピーします。**GET** ステートメントの構文を次に示します。

GET [STEP] (*x1!, y1!*) - **[STEP]**(*x2!,y 2!*), *arrayname*

引数 (*x1!, y1!*) と (*x2!, y2!*) には、イメージを囲む四角形の左上隅と右下隅の座標を指定します。引数 *arrayname* には、任意の数値配列を指定します。この配列は、**DIM** ステートメントで宣言します。指定する配列の大きさは、次の 3 つの要素で決まります。

- イメージを囲む四角形の高さと幅
- グラフィックスを出力するスクリーン モード
- 配列の型 (整数，長整数，単精度，倍精度)

注　意　イメージは、任意の数値型を持つ配列に保存することができますが、できる限り整数型の配列を使うようにしてください。整数型の配列を使うと、画面上のグラフィックス パターンはすべて整数で表現されます。単精度型や倍精度型の配列を使った場合、グラフィックス パターンによっては正確な実数で表すことができなくなるため、予測できない結果になる場合があります。

---

arrayname の大きさをバイト単位で計算するには、次の式を使ってください。

ByteSize = 4 + *planes* * **INT**(((*x2* - *x1* + 1) + 7) / 8) * ((*y2* - *y1*) +1)

planese は、画面表示に必要なプレーン数を示します。PC-9801 シリーズでは 4 です。*x1*, *x2* はイメージの左端，右端の x 座標、*y1*, *y2* は上端，下端の y 座標を示します。

配列要素の数を計算するには、上で求めた *ByteSize* を、配列内の 1 つの要素のバイト数で割ってください。このとき、配列のデータ型が重要になります。配列が整数型であるとき、各要素は 2 バイト (整数のサイズ) で表されます。したがって、実際の配列の大きさを求めるには、*ByteSize* を 2 で割らなければなりません。同じように、配列が長整数型なら、*ByteSize* を 4 で割ります (長整数は 4 バイト)。

次に、左上隅の座標が (10, 40)、右下隅の座標が (90, 80) の四角形の領域を保存するために必要な整数配列の大きさを計算する方法を示します。

1. まず、四角形の高さと幅を計算します。

```
RectHeight = ABS(y2 - y1) + 1 = 80 - 40 + 1 = 41
RectWidth  = ABS(x2 - x1) + 1 = 90 - 10 + 1 = 81
```

   y2 から y1 を引くとき、また x2 から x1 を引くときに、1 を加えることを忘れないでください。例えば、x1 = 10, x2 = 20 であるとき、その長方形の幅は 20 - 10 + 1 = 11 になります。

2. この整数配列の大きさのバイト数を計算します。

```
ByteSize = 4 + 4 * INT((RectWidth + 7) / 8) * RectHeight
         = 4 + 4 * INT((81 + 7) / 8) * 41
         = 4 + 4 * 11 * 41
         = 4 + 44 * 41
         = 1808
```

3. 前の手順で求めたバイト数を、配列要素当たりのバイト数 (整数の場合は 2) で割り、その結果を整数の近似値に丸めます。

```
1808 / 2 = 904
```

この配列の名前を Image とすると、次の **DIM** ステートメントによって、四角形のピクセル情報をコピーするために十分な大きさの配列が確保されます。

```
DIM Image (1 TO 904) AS INTEGER
```

注　意　**GET** ステートメントは、**WINDOW** ステートメントの後ではビューポート座標を使いますが、**GET** ステートメントに使う配列の大きさを計算するときには、物理座標を使わなければなりません (**WINDOW** ステートメントと、ビューポート座標から物理座標への変換方法については、「ビューポート座標の再定義」を参照してください)。

上の手順を使うと、イメージを保存するために最低限必要な配列のサイズを計算することができますが、これよりも大きなサイズの配列を宣言してもかまいません。例えば、次のステートメントも有効に機能します。

```
DIM Image (1 TO 1000) AS INTEGER
```

次のプログラムは、はじめに1つの楕円を描き、その内部を色で塗りつぶします。次に、**GET** ステートメントでその楕円を囲む四角形の領域をメモリにコピーします(楕円を別の位置に再表示するには、**PUT** ステートメントを使います。詳しくは、次の「イメージの移動: PUT」を参照してください)。

```
'配列を，長方形に納めるのに十分な大きさに設定します．
DIM Image (1 TO 1000) AS INTEGER

'長方形の内部に楕円を描きます．輪郭および内部を白で塗りつぶします．
CIRCLE (50, 60), 40, 2, , , .5
PAINT (50, 60), 7, 2

'長方形のイメージを配列に保存します．
GET (10, 40) - (90, 80), Image
```

イメージの移動: PUT

GET ステートメントはイメージのコピーをメモリに保存しますが、そのイメージを画面上の任意の位置に再表示するのが **PUT** ステートメントです。次に **PUT** ステートメントの構文を示します。

PUT (*x!*, *y!*), *arrayname* [, *actionverb*]

PUT ステートメントは、*arrayname* に保存されている四角形のイメージを、(*x!*, *y!*) で指定した位置を左上隅とする領域に再表示します。**PUT** で指定する座標は、1組だけです。

注　意　**WINDOW** ステートメントを **PUT** ステートメントの前で使うと、座標 (*x!*, *y!*) は四角形の左下隅を指定することになります。ただし、**SCREEN** キーワードを指定した場合は除きます。**WINDOW SCREEN** の後で **PUT** を使うと、(*x!*, *y!*) は長方形の左上隅の座標を示します。

例えば、次に示すステートメントを前のプログラムに加えると、**CIRCLE** や **PAINT** ステートメントを使って同じ図形を再び描いて塗り直すよりも速く、楕円を画面の右側に移動することができます。

```
PUT (200, 40), Image
```

ただし、次のステートメントのように、イメージが画面やビューポートの外にはみ出すような座標を指定しないように注意してください。

```
SCREEN 1
   .
   .
   .
'長方形の大きさは 141 ピクセル x 91 ピクセルです．
GET (10, 10)-(150, 100), Image
PUT (510, 120), Image
```

LINE や **CIRCLE** のような他のグラフィックス ステートメントと異なり、**PUT** ステートメントの出力は、ビューポートの境界で切り取られません。このエラーが起きると、“引数が有効範囲にありません” (ERR=5) というエラー メッセージが表示されます。

PUT ステートメントが持つもう 1 つの機能は、保存されたイメージと、それを再表示する位置に現在あるイメージとが、どのような関係になるかを制御できることです。この関係は、引数 *actionverb* で指定します。引数 *actionverb* には、**PSET, PRESET, AND, OR, XOR** のいずれかのキーワードが指定できます。

PSET キーワードを指定すると、再表示する位置の背景に関係なく (影響を受けることなく)、保存されたイメージをその位置にコピーすることができます。この場合、保存されたイメージを画面にそのままコピーするため、画面上にすでにあるイメージに重ね書きします。

表 2.1 は、**PUT** ステートメントで各キーワードを指定したときの、保存されたイメージのピクセルと画面上のピクセルの関係をまとめたものです。表の中で、1 はピクセルが ON であることを表し、0 はピクセルが OFF であることを表します。

表 2.1 PUT ステートメントの各キーワードの効果

キーワード	保存されたイメージ中のピクセル	PUT を実行する前の画面上のピクセル	PUT を実行した後の画面上のピクセル
PSET	0	0	0
	0	1	0
	1	0	1
	1	1	1
PRESET	0	0	1
	0	1	1
	1	0	0
	1	1	0
AND	0	0	0
	0	1	0
	1	0	0
	1	1	1
OR	0	0	0
	0	1	1
	1	0	1
	1	1	1
XOR	0	0	0
	0	1	1
	1	0	1
	1	1	0

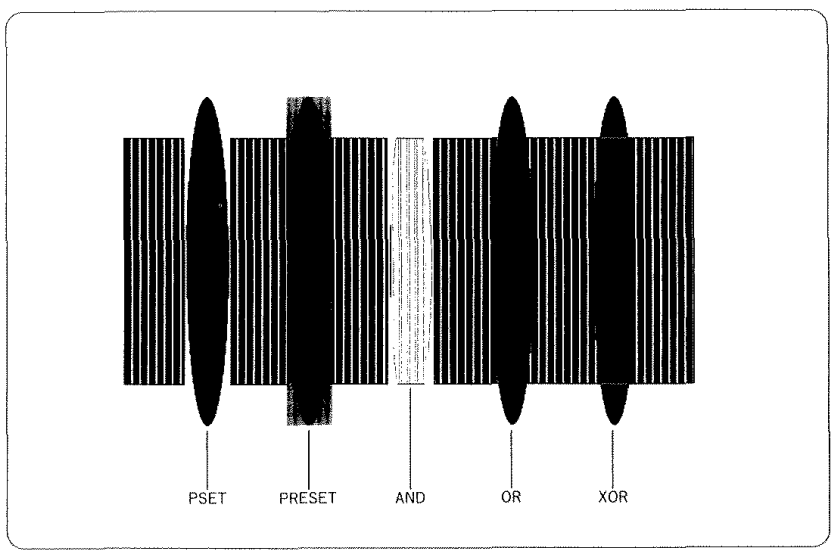
上の表からわかるように、これらのキーワードを使うと、ピクセルに対して論理演算が行われます。**PRESET** は、論理演算子 **NOT** と同様に、画面上のピクセルに関係なく、保存されたイメージ中のピクセルを反転させます。**AND, OR, XOR** は、同じ名前の論理演算子と同じ働きをします。例えば、次の論理演算を行うと結果は 0 になります。

```
1 XOR 1
```

これと同様に、**XOR** を指定すると、イメージ中のピクセルと背景のピクセルがともに **ON** になったとき、結果のピクセルが **OFF** になります。次のプログラムは、塗りつぶした四角形の上に、同じイメージを 5 つのキーワードを使って重ね合わせます。

```
DIM CircImage (1 TO 5000) AS INTEGER
SCREEN 1

'楕円を描き，色を塗り，GET を使ってそのイメージを保存します．
CIRCLE (22, 200), 160, , , , 8
PAINT (22, 200)
GET (0, 40)-(44, 360), CircImage
CLS

'ボックスを描き，その内部を指定したパターンで塗りつぶします．
LINE (40, 80)-(600, 320), , B
Pattern$ = CHR$(126) + CHR$(0) + CHR$(126) + CHR$(126)
PAINT (50, 90), Pattern$
'保存した楕円のイメージを，5 種類の異なる方法でボックスに重ね合わせます．
PUT (100, 40), CircImage, PSET
PUT (200, 40), CircImage, PRESET
PUT (300, 40), CircImage, AND
PUT (400, 40), CircImage, OR
PUT (500, 40), CircImage, XOR
```

画面例



色を使ったイメージでは、**PSET, PRESET, AND, OR, XOR** の各キーワードを使って、さらに複雑な操作ができます。これは、色の表示が単にピクセルを ON/OFF するだけの操作ではないからです。ただし、この場合でも論理演算との相似はやはり成り立っています。例えば、画面上のピクセルがパレット番号 1 (青) で、保存したイメージのピクセルがパレット番号 2 (緑) のとき、**PUT** ステートメントを実行すると、ピクセルの色は表 2.2 に示すようになります。

表 2.2 色に対する各キーワードの効果

キーワード	保存されたイメージ中のピクセルの色	PUT を実行する前の画面上のピクセルの色	PUT を実行した後の画面上のピクセルの色
PSET	10 (緑)	01 (青)	10 (緑)
PRESET	10 (緑)	01 (青)	01 (青)
AND	10 (緑)	01 (青)	00 (黒)
OR	10 (緑)	01 (青)	11 (水)
XOR	10 (緑)	01 (青)	11 (水)

PALETTE ステートメントを使わないと、パレット番号 1 (2 進数で 01) は青、パレット番号 2 (2 進数で 10) は緑、パレット番号 3 (2 進数で 11) は水色になります。

GET と PUT を使ったアニメーション

GET と **PUT** の 2 つのステートメントを使うと、アニメーションを効果的に実行することができます。特に、**XOR** と **PSET** の 2 つのキーワードは、アニメーションに適しています。**PSET** を使うと、アニメーションを速く処理できますが、画面の背景が消えてしまいます。一方、**XOR** を使うと、図形を移動したあとで画面の背景を復元することができます。次に、**XOR** を使ったアニメーションの手順を示します。

1. **XOR** を使って、オブジェクト (保存したイメージ) を画面に表示します。
2. 次にオブジェクトを表示する位置を計算します。
3. 再び **XOR** を使い、現在の表示位置にもう一度オブジェクトを表示します。これでオブジェクトは画面から消え、元の背景が復元されます。
4. 手順 1 に戻り、新しい位置にオブジェクトを表示します。

これらの手順を繰り返し実行すると、背景を変更せずにオブジェクトを移動することができます。手順 4 から 1 までの時間を短くし、手順 1 から 3 までの時間を十分にとると、画面のちらつきを減らすことができます。複数のオブジェクトによるアニメーションを行うときは、すべてのオブジェクトを同時に 1 ステップずつ処理してください。

背景を保存する必要がないときは、**PSET** キーワードを使ってアニメーションを行う方法があります。この場合、**GET** ステートメントでイメージをコピーするときに、イメージを囲む境界もいっしょに保存します。この境界がオブジェクトの最大移動距離よりも大きいと、イメージを新しい位置に表示するとき、元の位置にあるイメージの跡を、この境界によって消去することができます。この方法は、オブジェクトの移動を 1 つ

の **PUT** ステートメントだけで行うため、前に述べた **XOR** を使う方法よりも処理が速くなります (ただし、移動するイメージのサイズは大きくなります)。

次の例は、**PUT** ステートメントの **PSET** キーワードを使い、ボックスの底辺と左右の辺でバウンドするボールを表示します。**GET** ステートメントで保存したボールのイメージによって、塗りつぶしたボックスが消去されることを確かめてください。

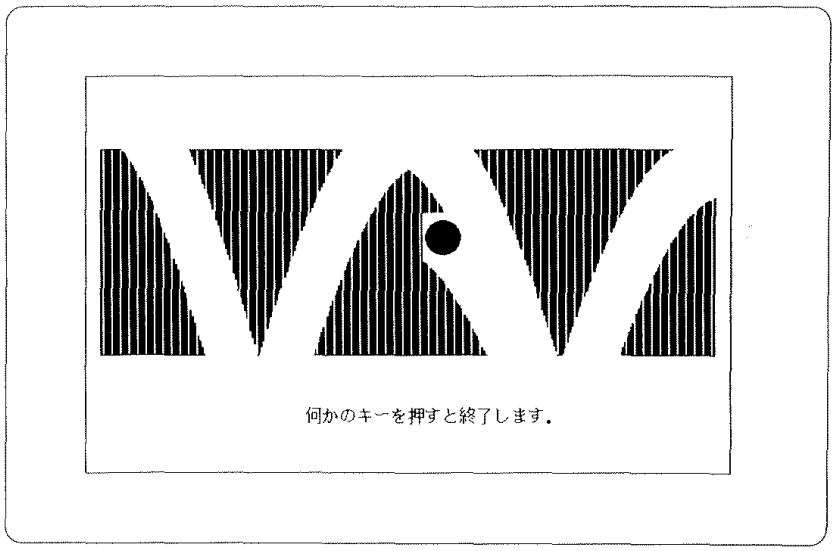
```
DECLARE FUNCTION GetArraySize (WLeft, WRight, WTop, WBottom)
SCREEN 1
'ビューポートを定義し，境界線を引きます．
VIEW (20, 10)-(620, 380), , 1

CONST PI = 3.141592653589#

'ビューポートの座標をビューポート座標によって再定義します．
WINDOW (-3.15, -.14)-(3.56, 1.01)

'プログラム中の配列を動的配列にします．
'$DYNAMIC

'CIRCLE と PAINT を使って，描画するイメージを囲む
'四角形のビューポート座標を計算します
WLeft = -.21
WRight = .21
WTop = .07
WBottom = -.07

'関数 GetArraySize を呼び出し，引数として
'イメージを囲む四角形の座標を渡します．
ArraySize% = GetArraySize(WLeft, WRight, WTop, WBottom)

DIM Array(1 TO ArraySize%)  AS INTEGER

'円を描いて塗りつぶします．
CIRCLE (0, 0), .18
PAINT (0, 0)

'イメージを囲む四角形を配列 Array に保存します．
GET (WLeft, WTop)-(WRight, WBottom), Array
CLS

'ボックスを描いて，特定のパターンで塗りつぶします．
LINE (-3, .8)-(3.4, .2), , B
Pattern$ = CHR$(126) + CHR$(0) + CHR$(126) + CHR$(126)
PAINT (0, .5), Pattern$

LOCATE 21, 29
PRINT "何かのキーを押すと終了します．"

'ループ変数を初期化します．
StepSize = .02
StartLoop = -PI
Decay = 1
```

```
DO
    EndLoop = -StartLoop
    FOR X = StartLoop TO EndLoop STEP StepSize

        'ボールがバウンドしたとき (ビューポートの底辺に当ったとき)
        '変数 Decay の値を小さくして, 次にバウンドするときの高さを
        '小さくします.
        Y = ABS(COS(X)) * Decay - .14
        IF Y < -.13 THEN Decay = Decay * .9

        'キーが押されるか, Decay が .01 より小さくなったら終了します.
        Esc$ = INKEY$
        IF Esc$ <> "" OR Decay < .01 THEN EXIT FOR

        '画面にイメージを表示します. 移動距離を表す StepSize は,
        'ボールを囲む境界よりも小さいため, イメージを移動するときに
        '直前の PUT で表示したイメージの跡を消去します (このとき,
        '画面上の背景も一緒に消去します).
        PUT (X, Y), Array, PSET
    NEXT X

    '方向を反転します.
    StepSize = -StepSize
    StartLoop = -StartLoop
LOOP UNTIL Esc$ <> "" OR Decay < .01

END
```

GetArraySize 関数は、ビューポート座標を物理座標に変換します。

```
FUNCTION GetArraySize (WLeft, WRight, WTop, WBottom)
    'パラメータとして渡されたビューポート座標を対応する物理座標に変換します.
    VLeft = PMAP(WLeft, 0)
    VRight = PMAP(WRight, 0)
    VTop = PMAP(WTop, 1)
    VBottom = PMAP(WBottom, 1)

    'イメージを囲む四角形の縦と横のピクセル数を計算します.
    RectHeight = ABS(VBottom - VTop) + 1
    RectWidth = ABS(VRight - VLeft) + 1

    '配列のバイトサイズを計算します.
    ByteSize = 4 + 4 * RectHeight * INT((RectWidth + 7) / 8)

    'Array は整数配列のため, 上で求めたバイト数を 2 で割ります.
    GetArraySize = ByteSize ¥ 2 + 1
END FUNCTION
```

画面例

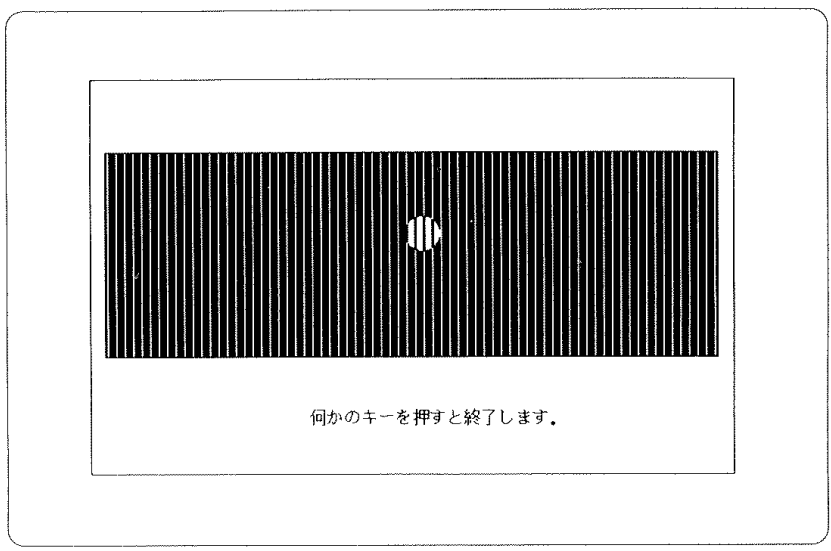


前のプログラムと次のプログラムを比較してください。次のプログラムは、最初に説明した4つの手順により、**PUT** ステートメントで **XOR** を指定して画面の背景を保存します。ボールを囲む四角形の境界を残す必要がないため、前のプログラムよりも四角形が小さくなっていることに注意してください。また、このプログラムでは、2つの **PUT** ステートメント (1つはイメージを表示し、もう1つはイメージを消去します) が必要です。2つの **PUT** ステートメントの間にある空の **FOR...NEXT** ループは、プログラムの実行を遅らせて、イメージの表示と消去を繰り返すために生じる画面のちらつきを減らします。

```
  .
  .
  .
'四角形の大きさが，前のプログラムより小さくなっています．そのため，配列の
'大きさも小さくなります．
WLeft = -.18
WRight = .18
WTop = .05
WBottom = -.05

DO
    EndLoop = -StartLoop
    FOR X = StartLoop TO EndLoop STEP StepSize
        Y = ABS(COS(X)) * Decay - .14

        '最初の PUT ステートメントでは，画面にイメージを表示します．
        PUT (X, Y), Array, XOR

        '空の FOR...NEXT ループを使って実行を遅らせることにより
        'イメージのちらつきを少なくします．
        FOR I = 1 TO 5: NEXT I
```

```
        IF Y < -.13 THEN Decay = Decay * .9
        Esc$ = INKEY$
        IF Esc$ <> "" OR Decay < .01 THEN EXIT FOR

        '2 番目の PUT ステートメントでは,
        'イメージを消去して, 背景を元に戻します.
        PUT (X, Y), Array, XOR
    NEXT X

    StepSize = -StepSize
    StartLoop = -StartLoop
LOOP UNTIL Esc$ <> "" OR Decay < .01
```

画面例



2.11 アプリケーションの例

ここでは、アプリケーションの例として、棒グラフを作成するプログラム、フラクタル図形の1つのマンデルブロート集合の点をいろいろな色を使って描画するプログラム、さらに、パターン エディタを説明します。

棒グラフの作成

棒グラフの作成プログラムは、すべての種類の **LINE** ステートメントを使って棒グラフを描きます。それぞれの棒は、**PAINT** ステートメントで指定したパターンで塗りつぶします。プログラムの入力データとして、グラフのタイトル，x 軸と y 軸のラベル，最大 5 つの棒とラベルの値を入力します。

このプログラムは、次のグラフィックス ステートメントを利用します。

- **LINE**
- **PAINT** (塗りつぶしパターンを指定)
- **SCREEN**

次に、棒グラフを生成するプログラム BAR.BAS を示します。

```
'タイトルのデータ型を定義します.
TYPE TitleType
    MainTitle AS STRING * 40
    XTitle AS STRING * 40
    YTitle AS STRING * 18
END TYPE

DECLARE SUB InputTitles (T AS TitleType)
DECLARE FUNCTION DrawGraph$ (T AS TitleType, Label$(),
  ⇒  Value!(), N%)
DECLARE FUNCTION InputData% (Label$(), Value!())

'タイトルと棒データを表す変数を宣言します.
DIM Titles AS TitleType, Label$(1 TO 5), Value(1 TO 5)

CONST FALSE = 0, TRUE = NOT FALSE
CONST BestMode = 88
CONST TextMode = 0
DO
    InputTitles Titles
    N% = InputData%(Label$(), Value())
    IF N% <> FALSE THEN
        NewGraph$ = DrawGraph$(Titles, Label$(), Value(), N%)
    END IF
LOOP WHILE NewGraph$ = "Y"

END
```

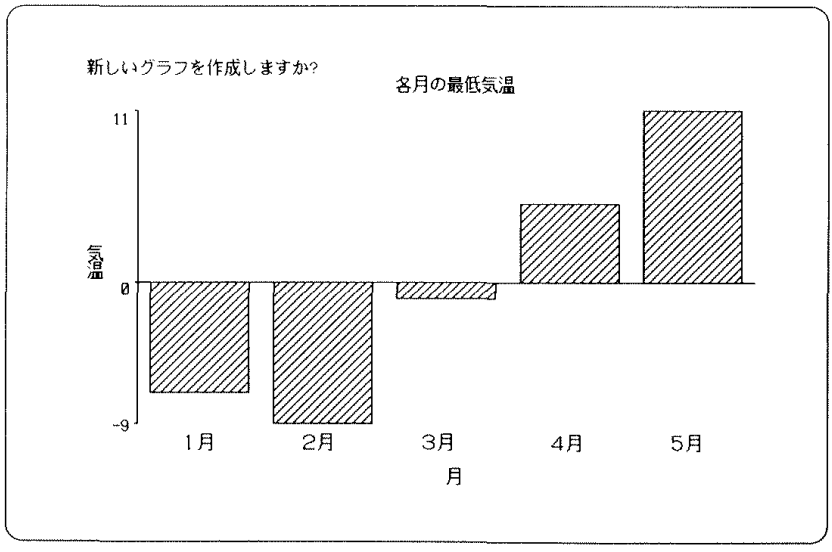
```
' ======================== DRAWGRAPH =======================
'   INPUTTITLES および INPUTDATA プロシージャで入力した
'   データから棒グラフを描きます.
' ==========================================================
STATIC FUNCTION DrawGraph$ (T AS TitleType, Label$(), Value(),
  ⇒  N%)

    'グラフのサイズを設定します.
    CONST GRAPHTOP = 48, GRAPHBOTTOM = 342
    CONST GRAPHLEFT = 48, GRAPHRIGHT = 624
    CONST YLENGTH = GRAPHBOTTOM - GRAPHTOP

    '最大値と最小値を計算します.
    YMax = 0
    YMin = 0
    FOR i% = 1 TO N%
        IF Value(i%) < YMin THEN YMin = Value(i%)
        IF Value(i%) > YMax THEN YMax = Value(i%)
    NEXT i%

    '棒の幅と各棒の間隔を計算します.
    BarWidth = (GRAPHRIGHT - GRAPHLEFT) / N%
    BarSpace = .2 * BarWidth
    BarWidth = BarWidth - BarSpace

    SCREEN BestMode
    CLS

    'y 軸を描きます.
    LINE (GRAPHLEFT, GRAPHTOP)-(GRAPHLEFT, GRAPHBOTTOM), 1

    'グラフのタイトルを表示します.
    Start% = 44 - (LEN(RTRIM$(T.MainTitle)) / 2)
    LOCATE 2, Start%
    PRINT RTRIM$(T.MainTitle);

    'y 軸のタイトルを表示します.
    Start% = CINT(13 - KLEN(RTRIM$(T.YTitle)) / 2)
    FOR i% = 1 TO KLEN(RTRIM$(T.YTitle))
        LOCATE Start% + i% - 1, 1
        PRINT KMID$(T.YTitle, i%, 1);
    NEXT i%

    'ラベルが 4 桁よりも大きくならないように, 縮小率を計算します.
    IF ABS(YMax) > ABS(YMin) THEN
        Power = YMax
    ELSE
        Power = YMin
    END IF
    Power = CINT(LOG(ABS(Power) / 100) / LOG(10))
    IF Power < 0 THEN Power = 0
```

```
'最小値と最大値を縮小します.
ScaleFactor = 10 ^ Power
YMax = CINT(YMax / ScaleFactor)
YMin = CINT(YMin / ScaleFactor)

'Power (べき乗の数) が 0 でなければ, 縮小率をグラフ上に表示します.
IF Power <> 0 THEN
    LOCATE 3, 2
    PRINT "x 10^"; LTRIM$(STR$(Power))
END IF

'y 軸の最大点の目盛と数値を表示します.
LINE (GRAPHLEFT - 3, GRAPHTOP)-STEP(3, 0)
LOCATE 4, 2
PRINT USING "####"; YMax

'y 軸の最小点の目盛と数値を表示します.
LINE (GRAPHLEFT - 3, GRAPHBOTTOM)-STEP(3, 0)
LOCATE 22, 2
PRINT USING "####"; YMin

YMax = YMax * ScaleFactor 'グラフに描画するために, 最小値と最大値を
YMin = YMin * ScaleFactor '元の値に戻します.

'x 軸のタイトルを表示します.
Start% = 44 - (LEN(RTRIM$(T.XTitle)) / 2)
LOCATE 25, Start%
PRINT RTRIM$(T.XTitle);

'y 軸のピクセル範囲を計算します.
YRange = YMax - YMin

'斜めの縞模様を定義します.
Tile$ = ""
FOR i% = 1 TO 8
    Tile% = 2 ^ (i% - 1)
    Tile$ = Tile$ + STRING$(4, Tile%)
NEXT
'必要に応じて, y=0 の線を引きます.
IF YMin < 0 THEN
    Bottom = GRAPHBOTTOM - ((-YMin) / YRange * YLENGTH)
    LOCATE INT((Bottom - 1) / 16) + 1, 5
    PRINT "0";
ELSE
    Bottom = GRAPHBOTTOM
END IF

'x 軸を描きます.
LINE (GRAPHLEFT - 3, Bottom)-(GRAPHRIGHT, Bottom)

'棒とラベルを描きます.
Start% = GRAPHLEFT + (BarSpace / 2)
FOR i% = 1 TO N%

    '棒のラベルを表示します.
    BarMid = Start% + (BarWidth / 2)
```

```
        CharMid = INT((BarMid - 1) / 8) + 1
        LOCATE 23, CharMid - INT(LEN(RTRIM$(Label$(i%))) / 2)
        PRINT Label$(i%);

        '棒を描き，縞模様で塗りつぶします．
        BarHeight = (Value(i%) / YRange) * YLENGTH
        LINE (Start%, Bottom)-STEP(BarWidth, -BarHeight), , B
        PAINT (BarMid, Bottom - (BarHeight / 2)), Tile$, 7

        Start% = Start% + BarWidth + BarSpace
    NEXT i%
    LOCATE 1, 1
    PRINT "新しいグラフを作成しますか? ";
    DrawGraph$ = UCASE$(INPUT$(1))

END FUNCTION

' ======================== INPUTDATA =======================
'                    棒のラベルと値を入力します
' ==========================================================
STATIC FUNCTION InputData% (Label$(), Value())

    'データ数を初期化します．
    NumData% = 0

    'データの入力を求めるメッセージを表示します．
    CLS
    PRINT "最大5つの棒のデータを入力してください:"
    PRINT "    * それぞれの棒のラベルと値を入力します．"
    PRINT "    * 値は負の数でもかまいません．"
    PRINT "    * 入力を終了するときは，空白のラベルを入力します．"
    PRINT
    PRINT "グラフの表示が終ったら，何かのキーを押して";
    PRINT "プログラムを終了します．"

    '空のラベルか 5 つのデータが入力されるまで，データ入力を行います．
    Done% = FALSE
    DO
        NumData% = NumData% + 1
        PRINT
        PRINT "Bar("; LTRIM$(STR$(NumData%)); "):"
        INPUT ; "          ラベル? ", Label$(NumData%)

        'ラベルが空白でないときだけ，値を入力します．
        IF Label$(NumData%) <> "" THEN
            LOCATE , 35
            INPUT "値? ", Value(NumData%)

            'ラベルが空白のときは，データカウンタを 1 つ減らして，
            'Done フラグに TRUE をセットします．
        ELSE
            NumData% = NumData% - 1
            Done% = TRUE
        END IF
    LOOP UNTIL (NumData% = 5) OR Done%
```

```
    '入力されたデータの数を返します.
    InputData% = NumData%

END FUNCTION

' ====================== INPUTTITLES ======================
'            グラフの 3 種類のタイトルを入力します
' ==========================================================
STATIC SUB InputTitles (T AS TitleType)
    SCREEN TextMode          'テキスト モードにセット
    DO                       'タイトル入力
        CLS
        INPUT "グラフのタイトルを入力してください: ", T.MainTitle
        INPUT "x軸のタイトルを入力してください    : ", T.XTitle
        INPUT "y軸のタイトルを入力してください    : ", T.YTitle

        'タイトルを変更するかどうかを確認します.
        LOCATE 7, 1
        PRINT "これでよろしいですか (Yで継続し, Nで変更します)? ";
        LOCATE , , 1
        OK$ = UCASE$(INPUT$(1))
    LOOP UNTIL OK$ = "Y"
END SUB
```

画面例



フラクタルのグラフィックス

このプログラムは、グラフィックス ステートメントを使って、“フラクタル”と呼ばれる図形を作成します。フラクタル図形は、数値演算の繰り返しによって発生する値をグラフィックスで表現したものです。このプログラムで生成するフラクタル図形は、複素数の一部の集合を表すものです。この集合は、米国 IBM 社の Thomas J. Watson 研究所の Benoit B. Mandelbrot 氏の名前にちなんで“マンデルブロート集合”と呼ばれています。

複素数は、実部と虚部の 2 つの部分で表されます。複素数の 2 乗を計算し、この実部と虚部によって次の複素数を求め、さらにこの複素数の 2 乗を計算します。このような計算を繰り返すと、ある複素数は急速に増大し別の値は安定した値となります。これらの安定値はマンデルブロート集合に含まれ、このプログラムでは黒で表現されます。マンデルブロート集合に含まれない不安定な値は、パレット中の他の色で表現されます。色のパレット番号が小さくなると、不安定な点となります。

プログラムは、1 行の描画が終るたびに、**PALETTE USING** ステートメントを使ってパレット中の色を回転させます。

このプログラムは、次のグラフィックス ステートメントを利用します。

- **LINE**
- **PALETTE USING**
- **PMAP**
- **PSET**
- **SCREEN**
- **VIEW**
- **WINDOW**

次に、マンデルブロート曲線を描画するプログラム MANDEL.BAS を示します。

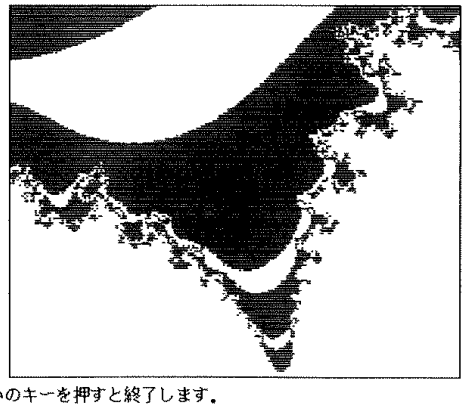
```
DEFINT A-Z              'デフォルトの変数型は整数です.

CONST BestMode = 88

DECLARE SUB ShiftPalette ()
DECLARE SUB WindowVals (WL%, WR%, WT%, WB%)
DECLARE SUB ScreenSet (EM%, CR%, VL%, VR%, VT%, VB%)

CONST FALSE = 0, TRUE = NOT FALSE   '論理定数を定義します.

'各点ごとの繰り返し回数の最大値を設定します.
CONST MAXLOOP = 30, MAXSIZE = 1000000

DIM PaletteArray(15)

FOR I = 0 TO 15: PaletteArray(I) = I: NEXT I
```

```
'WindowVals を呼び出して，ウィンドウの座標を獲得します．
WindowVals WLeft, WRight, WTop, WBottom

'ScreenSet を呼び出して，ビューポートの座標を獲得します．
ScreenSet EgaMode, ColorRange, VLeft, VRight, VTop, VBottom

'ビューポートとそれに対応するウィンドウを定義します．
VIEW (VLeft, VTop)-(VRight, VBottom), 0, ColorRange
WINDOW (WLeft, WTop)-(WRight, WBottom)

LOCATE 24, 10: PRINT "何かのキーを押すと終了します．";

XLength = VRight - VLeft
YLength = VBottom - VTop
ColorWidth = MAXLOOP ¥ ColorRange

'ビューポート中の各ピクセルに対して計算を行い，
'マンデルブロート集合に含まれるかどうかを求めます．
FOR Y = 0 TO YLength          'ビューポート内のすべての行について，
                              '繰り返します．
    LogicY = PMAP(Y, 3)       'ビューポート座標におけるピクセルの y 座標
                              'を求めます．
    PSET (WLeft, LogicY)      'その行の最左端のピクセルをプロットします．
    OldColor = 0              'はじめに背景色を設定します．

    FOR X = 0 TO XLength      '行の中のすべてのピクセルについて，
                              '繰り返します．
        LogicX = PMAP(X, 2)   'ビューポート座標におけるピクセルの x 座標
                              'を求めます．
        MandelX& = LogicX
        MandelY& = LogicY

        'この点がマンデルブロート集合に含まれるかどうか計算によって求めます．
        FOR I = 1 TO MAXLOOP
            RealNum& = MandelX& * MandelX&
            ImagNum& = MandelY& * MandelY&
            IF (RealNum& + ImagNum&) >= MAXSIZE THEN EXIT FOR
            MandelY& = (MandelX& * MandelY&) ¥ 250 + LogicY
            MandelX& = (RealNum& - ImagNum&) ¥ 500 + LogicX
        NEXT I

        'この点の色を割り当てます．
        PColor = I ¥ ColorWidth

        '色が前に描画した色と変った場合は，最後に参照した点から新しい点まで，
        '前の色を使って線を引きます．
        IF PColor <> OldColor THEN
            LINE -(LogicX, LogicY), (ColorRange - OldColor)
            OldColor = PColor
        END IF

        IF INKEY$ <> "" THEN END
    NEXT X
```

```
    '最後の点からビューポートの右端まで線を引きます.
    LINE -(LogicX, LogicY), (ColorRange - OldColor)

    '各行を描画した後でパレットをシフトさせます.
    ShiftPalette
NEXT Y

DO
    '何かのキーが押されるまで, パレットをシフトし続けます.
    ShiftPalette
LOOP WHILE INKEY$ = ""

END

'======================= ScreenSet =====================
'SCREEN 88 ステートメントを実行します.
'
'さらに, ビューポートの座標 (VL=左端, VR=右端, VT=上端, VB=下端) を
'設定し, ビューポートが完全な正方形になるように, 画面の縦横比を調整します.
'==============================================================
STATIC SUB ScreenSet (EM, CR, VL, VR, VT, VB)
    SCREEN BestMode
    EM = TRUE
    CLS
    VL = 110: VR = 529
    VT = 10: VB = 358
    CR = 15     '16色 (0-15) を使います.
END SUB

'====================== ShiftPalette ====================
'    呼び出されるたびに, パレットを 1 つずつ回転させます.
'==============================================================
STATIC SUB ShiftPalette
    SHARED PaletteArray(), ColorRange

    FOR I = 1 TO ColorRange
        PaletteArray(I) = (PaletteArray(I) MOD ColorRange) + 1
    NEXT I
    PALETTE USING PaletteArray(0)

END SUB

'======================= WindowVals =====================
'     ウィンドウの座標を入力します.
'     入力がないときは, デフォルトの値を設定します.
'==============================================================
STATIC SUB WindowVals (WL, WR, WT, WB)
    CLS
    PRINT "このプログラムは, マンデルブロート集合を"
    PRINT "グラフィックスで表現します. デフォルトの"
    PRINT "ウィンドウは, (-1000,625)-(250,-625)です. "
    PRINT "図の一部を拡大したいときは, ウィンドウの"
    PRINT "座標を入力してください. "
    PRINT
```

```
    PRINT "[ENTER] キーでデフォルトのウィンドウを設定し，"
    PRINT "他のキーでウィンドウの座標の入力を行います： ";
    LOCATE , , 1
    Resp$ = INPUT$(1)

    '[ENTER] 以外のキーが押されたら，ウィンドウの座標を入力します．
    IF Resp$ <> CHR$(13) THEN
        PRINT
        INPUT "左上隅のx座標： ", WL
        DO
            INPUT "右下隅のx座標： ", WR
            IF WR <= WL THEN
                PRINT "右端は左端よりも大きくなければなりません．"
            END IF
        LOOP WHILE WR <= WL
        INPUT "左上隅のy座標： ", WT
        DO
            INPUT "右下隅のy座標： ", WB
            IF WB >= WT THEN
                PRINT "下端は上端よりも小さくなければなりません．"
            END IF
        LOOP WHILE WB >= WT

    '[ENTER] が押されたら，デフォルトの値を設定します．
    ELSE
        WL = -1000
        WR = 250
        WT = 625
        WB = -625
    END IF
END SUB
```

画面例



何かのキーを押すと終了します．

パターン エディタ

このプログラムは、**PAINT** ステートメントで使うタイル パターンを編集します。画面の左側でタイルを編集しながら、画面の右側で完成したパターンをチェックすることができます。タイル パターンの編集が終了すると、プログラムはタイルの各行を **CHR$** の形式で表示します。

このプログラムは、次のグラフィックス ステートメントを利用します。

- **LINE**
- **PAINT** (塗りつぶしパターンを指定)
- **VIEW**

次に、パターンを編集するプログラム EDPAT.BAS を示します。

```
DECLARE SUB DrawPattern ()
DECLARE SUB EditPattern ()
DECLARE SUB Initialize ()
DECLARE SUB ShowPattern (OK$)

DIM Bit%(0 TO 7), Pattern$, Esc$, PatternSize%

DO
    Initialize
    EditPattern
    ShowPattern OK$
LOOP WHILE OK$ = "Y"

END

'======================= DRAWPATTERN ====================
'  画面の右側に四角形を描き，あるパターンで塗りつぶします
'=========================================================
STATIC SUB DrawPattern
    SHARED Pattern$

    VIEW (320, 48)-(622, 320), 0, 1 'ビューポートを設定します．
    PAINT (1, 1), Pattern$          'PAINT を使って塗りつぶします．
    VIEW                            'ビューポートを画面全体に戻します．

END SUB

'======================= EDITPATTERN ====================
'                    タイルパターンを編集します
'=========================================================
STATIC SUB EditPattern
    SHARED Pattern$, Esc$, Bit%(), PatternSize%

    ByteNum% = 1     '開始位置を指定します．
    BitNum% = 7
    Null$ = CHR$(0) 'CHR$(0)は，[↑] や [↓] が
                    '押されたときに返される，2バイトの文字列の
                    '先頭バイトを表します．
```

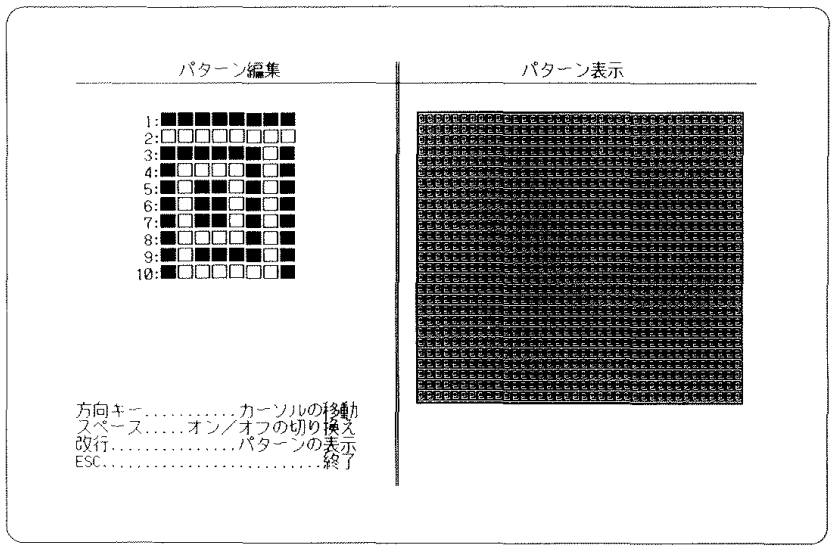
```
DO

    'このビットの画面上の位置を計算します.
    X% = ((7 - BitNum%) * 16) + 80
    Y% = (ByteNum% + 2) * 16

    'キーが押されるのを待ちます（カーソルを 1秒ごとに点滅させます）.
    State% = 0
    RefTime = 0
    DO

        'タイマーをチェックし, 1秒経過したらカーソルを反転します.
        IF ABS(TIMER - RefTime) > 1 THEN
            RefTime = TIMER
            State% = 7 - State%

            'ビットの境界のオン/オフを切り換えます.
            LINE (X% - 1, Y% - 1)-STEP(15, 14), State%, B
        END IF

        Check$ = INKEY$      'キーが押されたかどうかを調べます.

    LOOP WHILE Check$ = ""  '何かのキーが押されるまでループします.

    'カーソルを消去します.
    LINE (X% - 1, Y% - 1)-STEP(15, 14), 0, B

    SELECT CASE Check$  'キーの処理を行います.

    CASE CHR$(27)        '[ESC] キーが押された場合.
        EXIT SUB         'SUBプロシージャを終了します.

    CASE CHR$(32)        '[SPACE] キーが押された場合.
                         'ビットの状態をリセットします.
        'パターン文字列中のビットを反転します.
        CurrentByte% = ASC(MID$(Pattern$, ByteNum% * 4, 1))
        CurrentByte% = CurrentByte% XOR Bit%(BitNum%)
        MID$(Pattern$, ByteNum% * 4) = CHR$(CurrentByte%)
        MID$(Pattern$, ByteNum% * 4 - 1)
          ⇒ = CHR$(CurrentByte%)
        MID$(Pattern$, ByteNum% * 4 - 2)
          ⇒ = CHR$(CurrentByte%)
        MID$(Pattern$, ByteNum% * 4 - 3)
          ⇒ = CHR$(CurrentByte%)

        '画面上のビットを描きなおします.
        IF (CurrentByte% AND Bit%(BitNum%)) <> 0 THEN
            CurrentColor% = 1
        ELSE
            CurrentColor% = 0
        END IF
        LINE (X% + 1, Y% + 1)-STEP(11, 10), CurrentColor%,
          ⇒ BF
```

```
        CASE CHR$(13)          '[ENTER] が押された場合.
            DrawPattern        '左側のボックスにパターンを描きます.

        CASE Null$ + CHR$(75)   '[←] キー: カーソルを左に移動します.
            BitNum% = BitNum% + 1
            IF BitNum% > 7 THEN BitNum% = 0

        CASE Null$ + CHR$(77)   '[→] キー: カーソルを右に移動します.
            BitNum% = BitNum% - 1
            IF BitNum% < 0 THEN BitNum% = 7

        CASE Null$ + CHR$(72)   '[↑] キー: カーソルを上に移動します.
            ByteNum% = ByteNum% - 1
            IF ByteNum% < 1 THEN ByteNum% = PatternSize%

        CASE Null$ + CHR$(80)   '[↓] キー: カーソルを下に移動します.
            ByteNum% = ByteNum% + 1
            IF ByteNum% > PatternSize% THEN ByteNum% = 1

        CASE ELSE
            '[ESC], [SPACE], [ENTER], [↑], [↓], [←], [→]
            '以外のキーが押された場合は, 何も行いません.
        END SELECT
    LOOP
END SUB

'======================= INITIALIZE =====================
'                初期パターンと画面を設定します
'========================================================
STATIC SUB Initialize
    SHARED Pattern$, Esc$, Bit%(), PatternSize%

    Esc$ = CHR$(27)      'ESC 文字は ASCII コードで 27 です.

    '配列にビット 0 からビット 7 までの値を格納します.
    FOR I% = 0 TO 7
        Bit%(I%) = 2 ^ I%
    NEXT I%

    CLS

    'パターンのサイズを (バイト数で) 入力します.
    LOCATE 5, 5
    PRINT "パターンサイズ (1-16行) を入力してください:";
    DO
        LOCATE 5, 50
        PRINT "  ";
        LOCATE 5, 50
        INPUT "", PatternSize%
    LOOP WHILE PatternSize% < 1 OR PatternSize% > 16

    '初期パターンとしてすべてのビットをセットします.
    Pattern$ = STRING$(PatternSize% * 4, 255)
```

```
    SCREEN 88
    CLS

    '線を引いて画面を分割します.
    LINE (0, 20)-(635, 20), 1
    LINE (300, 0)-(300, 398)
    LINE (302, 0)-(302, 398)

    'タイトルを表示します.
    LOCATE 1, 13: PRINT "パターン編集"
    LOCATE 1, 53: PRINT "パターン表示"

    'パターンの編集画面を描きます.
    FOR I% = 1 TO PatternSize%

        '各行の左にラベルを表示します.
        LOCATE I% + 3, 8
        PRINT USING "##:"; I%

        '各“ビット”を表すボックスを描きます.
        X% = 80
        Y% = (I% + 2) * 16
        FOR J% = 1 TO 8
            LINE (X%, Y%)-STEP(13, 12), 1, BF
            X% = X% + 16
        NEXT J%
    NEXT I%

    DrawPattern      '“パターン表示”ボックスを描きます.

    LOCATE 21, 1
    PRINT "方向キー...........カーソルの移動"
    PRINT "SPACE........オン／オフの切り換え"
    PRINT "ENTER............パターンの表示"
    PRINT "ESC......................終了";

END SUB

'======================== SHOWPATTERN ====================
'   PAINT のパターンを作成するときの CHR$ の値を表示します
'=========================================================
STATIC SUB ShowPattern (OK$)
    SHARED Pattern$, PatternSize%

    '80桁のテキストモードに戻します.
    SCREEN 0
    CLS

    PRINT "次の文字列を使ってパターンを作成してください:"
    PRINT
```

```
    'パターンの各バイトの値を表示します.
    FOR I% = 1 TO PatternSize%
        PatternByte% = ASC(MID$(Pattern$, I% * 4 - 3, 1))
        PRINT "CHR$("; LTRIM$(STR$(PatternByte%)); ")"
    NEXT I%
    PRINT
    LOCATE , , 1
    PRINT "新しいパターンを作成しますか? ";
    OK$ = UCASE$(INPUT$(1))
END SUB
```

画面例



3 QB4.5 および BC7.1 との互換性

Microsoft Visual Basic for MS-DOS (VBDOS) は、QuickBasic バージョン 4.5 (QB4.5) に対する上位互換性を持ち、QB4.5 のすべての予約語をサポートしています。また一部の機能を除いて、Microsoft Basic Professional Development System バージョン 7.1 (BC7.1) との互換性があります。ここでは、VBDOS と QB4.5 および BC7.1 の機能的な相違について、キーワードごとに説明します。

3.1 予約語の互換性

ここでは、BC7.1 で追加された命令やキーワード，VBDOS で追加された命令やキーワード，BC7.1 で追加されたが VBDOS ではサポートされていない命令に分け、一覧表を示します。VBDOS の予約語の一覧については、『リファレンス』の付録 A を参照してください。また、各命令の機能については、『リファレンス』またはオンラインヘルプを参照してください。* が付いている命令は、**PRINT** ステートメントに対する **PRINT** メソッドのように、予約語としては以前から存在していたが、同じキーワードで別の命令が設けられたことを示します。

BC7.1 と VBDOS でのみ有効な命令

CCUR 関数
CURDIR$ 関数
CVC 関数
DIR$ 関数
MKC$ 関数
SSEG 関数
STACK 関数
ERR ステートメント (関数) *
CHDRIVE ステートメント
CURRENCY キーワード
DEFCUR ステートメント
EVENT ステートメント
PRESERVE キーワード (**REDIM**)
SSEGADD 関数
STACK ステートメント

VBDOS でのみ有効な命令

ADDITEM メソッド
CONTROL キーワード
DATESERIAL 関数
DOEVENTS 関数
ENDDOC メソッド
FORM キーワード
$FORM メタコマンド
CLIPBOARD オブジェクト
DATEVALUE 関数
DAY 関数
DRAG メソッド
ERROR$ 関数
FORMAT$ 関数
GETTEXT メソッド

HIDE メソッド
HOUR 関数
INPUTBOX$ 関数，ステートメント
LOAD ステートメント
MINUTE 関数
MONTH 関数
MOVE メソッド
MSGBOX 関数
MSGBOX ステートメント
NEWPAGE メソッド
NOW 関数
OPTION EXPLICIT ステートメント
PRINTER オブジェクト
PRINTFORM メソッド
QBCOLOR 関数
REFRESH メソッド
REMOVEITEM メソッド
RGB 関数
SECOND 関数
SETFOCUS メソッド
SETTEXT メソッド
SHOW メソッド
TEXTHEIGHT メソッド
TEXTWIDTH メソッド
TIMESERIAL 関数
TIMEVALUE 関数
TYPEOF キーワード
UNLOAD ステートメント
WEEKDAY 関数
YEAR 関数
CLEAR メソッド (ステートメント)*
CLS メソッド (ステートメント)*
PRINT メソッド (ステートメント)*
SCREEN オブジェクト (ステートメント，関数)

BC7.1 でのみ有効な命令

これらの命令は VBDOS ではサポートされていませんが、VBDOS でも予約語として扱われます。

ISAM 関連

ALL キーワード (**ROLLBACK ALL**)
BEGINTRANS ステートメント
BOF 関数
CHECKPOINT ステートメント
COMMITTRANS ステートメント
CREATEINDEX ステートメント
DELETE ステートメント
DELETEINDEX ステートメント
DELETETABLE ステートメント
GETINDEX$ 関数
INSERT ステートメント
ISAM キーワード (**OPEN**)
MOVEFIRST ステートメント
MOVELAST ステートメント
MOVENEXT ステートメント
MOVEPREVIOUS ステートメント
RETRIEVE ステートメント
ROLLBACK ステートメント
SAVEPOINT 関数
SEEKGT ステートメント
SEEKGE ステートメント
SEEKEQ ステートメント
SETINDEX ステートメント
TEXTCOMP 関数
UPDATE ステートメント

MS OS/2 関連

SIGNAL ステートメント
SHELL 関数

3.2 各製品の設計上の相違

各製品の主な設計上の相違点は次のとおりです。

- **バイナリ セーブのフォーマット**

  ファイルの保管時に、QB4.5 の [QuickBASIC 標準ファイル]，BC7.1 の [Binary - Fast load and save]，VBDOS の [バイナリ形式] の各オプションを選択すると、ファイルはバイナリ形式で保管されます。3 つの製品で、それぞれバイナリ形式のフォーマットが異なります。したがって、作成時と異なるプログラミング環境でファイルをロードするには、テキスト形式で保存しておく必要があります。

- **オブジェクト ファイルのフォーマット**

  各製品のコンパイラが生成するオブジェクト ファイルに記録される情報には、それぞれ相違があります。各製品間で、オブジェクト ファイル (.OBJ)，Quick ライブラリ (.QLB)，オブジェクト モジュール ライブラリ (.LIB) をそのまま流用することはできません。必ず、ソース ファイルを再コンパイルしてください。

- **浮動小数点演算ライブラリ**

  QB4.5 および BC7.1 は、エミュレータ ライブラリと代替数値演算ライブラリの両方をサポートしていますが、VBDOS はエミュレータ ライブラリのみをサポートしています。演算精度や実行スピードなどが異なります。

- **文字列データの格納位置**

  文字列データは、QB4.5 は near メモリのみ，BC7.1 は near メモリと far メモリの両方，VBDOS は far メモリのみに格納されます。

- **フォーム，コントロールおよびイベントドリブン プログラミング モデルの導入**

  VBDOS では、新しいユーザー インターフェイスとプログラミング モデルが採用されています。

- **MS OS/2 のサポート**

  BC7.1 では MS OS/2 用のアプリケーションの作成が可能でしたが、QB4.5 および VBDOS ではこの機能はサポートされていません。

3.3 各製品の主な仕様の相違

各製品の主な仕様の違いは次のとおりです。

■ スクリーン モード

用意されているスクリーン モードの種類と、デフォルトのスクリーン モードがそれぞれ異なります。QB4.5 ではスーパーインポーズ モード (88), BC7.1 ではテキスト モード (82, VBDOS の 82 とは異なります) がデフォルトです。VBDOS では、フォームを使うアプリケーションでは 82, 使わないアプリケーションでは 81 がデフォルトです。**SCREEN** ステートメントを実行していない BC7.1 のアプリケーションは、基本的に VBDOS でもそのまま動作します。**SCREEN** ステートメントを実行せずにグラフィックスの操作を行っている QB4.5 のアプリケーションは、**SCREEN** ステートメントで適切な設定を行わなければ VBDOS で動作しません。

■ 色の指定方法

テキスト画面の色の指定が、QB4.5 および BC7.1 は 8 色を基準に設計されていましたが、VBDOS は 16 色を基準に設計されています。色番号は、QB4.5 および BC7.1 は 0～7 ですが、VBDOS では 0～15 です。また、グラフィックス パレットのデフォルトの設定が、QB4.5 および BC7.1 では 明るい色がより小さなパレット番号に割り当てられていましたが、VBDOS ではより大きな番号に割り当てられています。

■ 画面の初期化のタイミング

画面の初期化が行われる条件が、各製品でそれぞれ異なります。VBDOS では、新しく追加された **SCREEN** ステートメントの 5 番目の引数である *noinit%* を使うと、画面の初期化の有無を制御できます。

■ キーボード上の各キーを示す値

BC7.1 では MS OS/2 を基準とした値、VBDOS では Microsoft Windows を基準とした値になっています。**KEY** ステートメントでキーを指定する値が異なります。

■ ファンクション キーの扱い

VBDOS では、ファンクション キーが 16 個あることを基準に設計されています。多い番号のファンクション キーが用意されていない機種では、指定しても単に無視されます。

■ 配列の EMS メモリへの格納

BC7.1 および VBDOS では、/Ea オプションを指定してプログラミング環境を起動すると、可変長文字列を除く 512 バイト～16K バイトの配列は、EMS メモリに格納されます。**VARSEG, VARPTR** で求めたアドレスに対して、**PEEK, POKE** などのメモリ操作を行っているプログラムでは、/Ea オプションは指定しないでください。誤動作の原因となります。

■ far メモリのアドレス

QB4.5 では、可変長文字列もデフォルトのデータ セグメントである DGROUP に格納されていたため、**SADD** 関数でオフセットアドレスを得るだけで、オブジェクトの位置を特定できました。BC7.1 で far 文字列を利用している場合および VBDOS では、セグメント アドレスも得る必要があります。セグメント アドレスを返す **SSEG** 関数か、セグメントとオフセットの両方を返す **SSEGADD** 関数を使ってください。また、**VARSEG** または **VARPTR** 関数を使っている場合は、プログラムの書き換えが必要です。文字列データのセグメントを得るために **VARSEG** 関数を使っているときは、**SSEG** 関数に置き換えてください。また、文字列記述子を操作するために **VARPTR** 関数を使っているときは、記述子の構造が far 文字列と near 文字列で異なるため、より大きな変更が必要です。

■ フォームとステートメント，関数

VBDOS では、フォームが表示されているときに、多くの入出力ステートメントおよび関数が実行できません。

3.4 ステートメントおよび関数の機能の互換性

一部のステートメントおよび関数は、QB4.5, BC7.1, VBDOS でそれぞれ仕様が変更されています。ここでは、アルファベット順に、主なステートメントおよび関数の仕様上の相違点について説明します。それぞれの機能の詳細は、『リファレンス』またはオンライン ヘルプを参照してください。

CDBL$, CSNG$ 関数

これらの関数は 1 バイト文字と 2 バイト文字を相互に変換しますが、QB4.5 では、BC7.1 および VBDOS と一部の変換結果が異なります。

1 バイト文字		BC7.1 および VBDOS		QB4.5	
'	(27)	’	(2147)	´	(212D)
<	(3C)	＜	(2163)	〈	(2152)
>	(3E)	＞	(2164)	〉	(2153)
~	(7E)	―	(2131)	～	(2141)

CLS ステートメント

BC7.1 では画面の背景色で塗りつぶされましたが、VBDOS では必ず黒で塗りつぶされます。

COLOR ステートメント

文字の色を指定する色番号は、QB4.5 および BC7.1 では 0～7 の値でしたが、VBDOS では 0～15 になっています。ただしスクリーン モード 0 (81), 2 (88), 84 では、実際に表示されるのは 8 色で、8～15 は 0～7 と同じ色を示します。この場合は、QB4.5 や BC7.1 のアプリケーションは、VBDOS でも同じように動作します。VBDOS のスクリーン モード 82 では、テキスト モードであっても背景をグラフィックス画面を使って表示するため、背景色は 16 色から選択できます。

BC7.1 では、背景色を示す *background* 引数を指定しても、*foreground* 引数に 0 以外が指定されていると、背景色は変更されませんでした (文字色優先)。VBDOS では、*background* 引数を指定すると、*foreground* 引数が無視されます (背景色優先)。

QB4.5 および BC7.1 では、グラフィックスまたはスーパーインポーズ モードで背景色を設定すると、パレット番号 0 の値が自動的に変更されていましたが、VBDOS では文字の背景色だけが変更され、パレットの設定には影響しません。

点滅，反転の指定方法がそれぞれ異なります。

FRE 関数

far データ領域，DGROUP，フォームに関するデータ セグメントなど、チェックの対象となる領域が一部異なります。

KEY ステートメント (代入)

VBDOS では、16 個のファンクション キーが指定可能です。実際のキーボードに存在しないキーは、指定しても単に無視されます。

KEY ステートメント (イベント トラッピング)

VBDOS では、キー イベントのトラップの設定に使う値が異なります。QB4.5, BC7.1 で 30 および 31 に割り当てられていた [STOP] および [END] ([HELP]) キーは、VBDOS ではユーザー定義キーとして定義しなければなりません。また、[SHIFT], [CTRL], [メニュー] ([GRPH]) キーの検出方法が変更になり、キーの組み合わせを無視する設定が可能になるとともに (*keyboardflag%*=128)、[英数カナ], [NUM LOCK], [CAPS LOCK] は検出できなくなっています。さらに、ユーザー定義キーの定義に使うキーボード スキャン コードが変更されています。

ON ERROR ステートメント

BC7.1 および VBDOS では、**LOCAL** キーワードを指定して、プロシージャ内だけに有効なローカルなエラー処理ルーチンを設定できます。

PCOPY ステートメント

VBDOS のプログラミング環境は、自分自身を表示するために、デフォルトでグラフィックス スクリーン ページを使用します (スクリーン モード 0 (88) および 1 (87) ではページ 1，84 ではページ 2 と 3)。これらのページを **PCOPY** ステートメントで指定すると、エラーが発生します。起動時に /F オプションを指定すると、QB4.5 および BC7.1 と同じ機能になります。

PRINT ステートメント

VBDOS では、ENTER や TAB などを除く ASCII コード &H0～&H1F, &H7F, &HFD～&HFF の制御コードに対して、罫線や記号などのキャラクタが割り当てられています。例えば QB4.5 および BC7.1 で PRINT CHR$(7) を実行するとビープ音がなりましたが、VBDOS で実行すると罫線キャラクタが表示されます。

RANDOMIZE ステートメント

QB4.5 および BC7.1 では、引数を省略して実行すると、乱数系列の初期化に使う値の入力を求めるメッセージが表示されます。VBDOS では、引数を省略すると、自動的に **TIMER** 関数の戻り値が使われます。

REDIM ステートメント

BC7.1 および VBDOS では、動的配列に対して **PRESERVE** キーワードを指定して **REDIM** ステートメントを実行すると (**REDIM PRESERVE**)、配列のサイズを変更しても、その配列にすでに格納されているデータをそのまま残すことができます。

SCREEN 関数

文字の表示色を得るときの戻り値の各ビットの意味が変更になっています。

SCREEN ステートメント

QB4.5 および BC7.1 ではスクリーン モード 81 と 82 は同じ意味でしたが、VBDOS ではスクリーン モード 82 の意味が異なります。VBDOS のスクリーン モード 82 では、文字はテキスト画面に表示されますが、文字の背景やフォームの一部はグラフィックス画面に表示されます。したがって、文字の背景色として 16 色利用できます。

スクリーン モード 0 は、QB4.5 は 88, BC7.1 は 81 および 82 と同じ意味でしたが、VBDOS では 81 と同じ意味です。また、**SCREEN** ステートメントで特に指定しないときのデフォルトのスクリーン モードは、QB4.5 および BC7.1 では 0 でしたが、VBDOS では、フォームを使っているアプリケーションは 82、使っていないアプリケーションは 0 (81) です。このデフォルトのモードが、QB4.5 ではスーパーインポーズ モードであるのに対し、BC7.1 および VBDOS はテキスト モードであることに注意してください。

VBDOS で、画面の初期化を制御するための 5 番目の引数 *noinit%* が導入されています。

2 番目の引数 *paletteswitch%* に 0 を指定したときの意味が、QB4.5 および BC7.1 では 8 色中 8 色表示でしたが、VBDOS では同時表示可能な最大色数に変更されています。VBDOS がサポートする PC-9801 シリーズでは、16 色を意味します。

グラフィックス パレットのデフォルトの設定が、QB4.5 および BC7.1 では小さい番号により明るい色が割り当てられていましたが、VBDOS ではより大きい番号に割り当てられています。

VBDOS のプログラミング環境は、自分自身を表示するために、デフォルトでグラフィックス スクリーン ページを 1 枚使用します。*apage%* および *vpage%* 引数でこれらのページを 指定すると、エラーが発生します。QB4.5 および BC7.1 で作成した、複数のグラフィックス スクリーン ページを使用しているアプリケーションを VBDOS で実行するには、起動時に /F オプションを指定してください。

QB4.5 の環境変数 QBGRPNEC および BC7.1 のスクリーンモード 89 は、VBDOS ではサポートされていません。

4 QB4.5 および BC7.1 アプリケーションの移植

この章では、QuickBasic バージョン 4.5 (QB4.5) または Microsoft Basic Professional Development System バージョン 7.1 (BC7.1) で作成されたアプリケーションを、Microsoft Visual Basic for MS-DOS に移植する方法について説明します。Visual Basic for MS-DOS との互換性が BC7.1 と QB4.5 で異なるときは、それぞれ明示的に示します。

4.1 一般的な考え方

QB4.5 で作成したアプリケーションは、第 3 章で説明した仕様上の変更点を除き、ほとんどそのまま Visual Basic for MS-DOS でも動作します。また、BC7.1 で作成したアプリケーションは、ISAM やプレゼンテーション グラフィックス，MS OS/2 のサポートなど、BC7.1 特有の機能を使っていなければ、同様にほとんどそのまま動作します。Visual Basic for MS-DOS は、扱えるプログラム サイズやデータ量を大きくできるように設計されており、アプリケーションの実行スピードの向上も図られていますから、特に QB4.5 で作成したアプリケーションは、開発環境を Visual Basic for MS-DOS に切り換えるだけで、多くのメリットを得ることができます。

Visual Basic for MS-DOS の最も大きな改良点は、フォームやコントロールなどを使ったグラフィカルなユーザー インターフェイスと、それに連動したイベントドリブン プログラミング モデルの採用です。Visual Basic for MS-DOS で新規にアプリケーションを開発する場合、Microsoft Windows に似た操作性を持つ、ユーザー インターフェイスに優れた MS-DOS アプリケーションを、比較的簡単に作成することができます。

しかし、QB4.5 または BC7.1 で作成したアプリケーションを、Visual Basic for MS-DOS の特長をフルに活かしたアプリケーションに改良するには、大幅な再設計が必要です。具体的には、ユーザーインターフェイスを再設計するとともに、アプリケーションの機能を入力，出力，内部的な処理に分けて構造化し、イベントドリブン プログラミング モデルに従った実行の流れに変更する必要があります。ユーザー インターフェイスに関する部分については再度プログラミングしなければなりません。例えば画面表示にグラフィックスを利用している場合、グラフィックスが表示されている状態ではフォームは利用できません。しかし、QB4.5 または BC7.1 で作成したアプリケーションがよく構造化されていれば、多くのプロシージャはそのまま利用できるはずです。本格的な Visual Basic for MS-DOS のアプリケーションを作成する方法については、『プログラミング ガイド』を参照してください。また、パッケージに含まれているサンプル アプリケーションを参考にして、イベントドリブン プログラミング モデルの実行の流れを追ってみてください。

ここでは、Quick Basic で作成したアプリケーションの基本的な構造は残して、ユーザーからの入力やユーザーに対するメッセージの表示にダイアログ ボックスを利用して、“Visual Basic らしさ” をアプリケーションに加える方法について説明します。

▼ QB4.5 または BC7.1 アプリケーションを Visual Basic へ移植するには

1. QB4.5 または BC7.1 の Quick ライブラリは、そのままでは Visual Basic for MS-DOS で利用できません。Visual Basic for MS-DOS 用にライブラリをビルドし直すか、Visual Basic for MS-DOS と互換性のあるライブラリを使用してください。
2. アプリケーションを Visual Basic for MS-DOS のプログラミング環境にロードします。QB4.5 または BC7.1 でソース ファイルをテキスト形式で保存してあれば、そのまま Visual Basic for MS-DOS のプログラミング環境にロードできます。バイナリ形式で保存されている場合は、いったん QB4.5 または BC7.1 でロードし、テキスト形式で保存しなおしてください。
3. 定数，変数，ユーザー定義関数，ユーザー定義型，ラベル，プロシージャなどの名前として、新しく追加された予約語と同じ名前を使っていないことを確認します。
4. 第 3 章を参考に仕様上の変更点を確認し、必要があれば修正します。
5. 必要に応じて、ユーザーインターフェイスを改良します。
6. アプリケーションをコンパイル，リンクします。

4.2 Quick ライブラリ (.QLB)

QB4.5 または BC7.1 の Quick ライブラリは、そのままでは Visual Basic for MS-DOS で利用できず、再度コンパイルとリンクを行う必要があります。また、near 文字列特有の処理を行っているルーチンは、Visual Basic for MS-DOS の far 文字列を使って書き直す必要があります。

一部の互換性は、Quick ライブラリの提供形態に依存します。お使いの Quick ライブラリには、QB4.5 または BC7.1 で提供されたもののほかに、Basic や他の言語で独自に作成したものもあるはずです。ここでは、そのそれぞれに対して、Quick ライブラリの互換性について説明します。

QB4.5 と BC7.1 で提供されている Quick ライブラリ

QB4.5 または BC7.1 で提供されている Quick ライブラリと Visual Basic for MS-DOS の Quick ライブラリの対応を次に表に示します。

QB4.5 の Quick ライブラリ/インクルード ファイル	Visual Basic for MS-DOS の Quick ライブラリ/インクルード ファイル
QB.QLB/QB.BI	VBDOS.QLB/VBDOS.BI

BC7.1 の Quick ライブラリ/インクルード ファイル	Visual Basic for MS-DOS の Quick ライブラリ/インクルード ファイル
QBX.QLB/QBX.BI	VBDOS.QLB/VBDOS.BI
CHRTBEFR.QLB/CHRTB.BI	なし
FONTBEFR.QLB/FONTB.BI	なし
MATBEFR.QLB/MATB.BI	なし
UITBEFR.QLB/GENERAL.BI, MENU.BI, MOUSE.BI, QBX.BI, WINDOW.BI	Visual Basic for MS-DOS では、ユーザー インターフェイス機能が統合されています。詳しくは、表の後の説明を参照してください。

UITBEFR.QLB (ユーザー インターフェイス ツールボックス) で提供される機能は、Visual Basic for MS-DOS ではユーザー インターフェイスを作成する標準の機能として提供されています。したがって、UITBEFR.QLB を使うアプリケーションは、Visual Basic for MS-DOS のフォームとコントロールを使って書き直す必要があります。

Basic で書かれた Quick ライブラリ

QB4.5 または BC7.1 で作成した Quick ライブラリは、そのままでは利用できません。Visual Basic for MS-DOS で提供されている Basic コンパイラ (BC.EXE) とリンカ (LINK.EXE) を使って、コンパイルとリンクを再度行ってください。新しく作成する Quick ライブラリは、VBDOSQLB.LIB とリンクします。Quick ライブラリの作成方法については、『プログラミングガイド』第 18 章を参照してください。

C で書かれた Quick ライブラリ

Microsoft C で作成した QB4.5 または BC7.1 用の Quick ライブラリは、そのままでは利用できません。Microsoft C/C++ バージョン 7.0A で再コンパイルし、Visual Basic for MS-DOS のリンカ (LINK.EXE) で再リンクする必要があります。新しく作成する Quick ライブラリは、VBDOSQLB.LIB とリンクします。Quick ライブラリの作成方法については、『プログラミングガイド』第 18 章を参照してください。

Quick ライブラリのコードで near 文字列のアドレスを使っているときは、far 文字列を使って書き直す必要があります。

MASM で書かれた Quick ライブラリ

Microsoft Macro Assembler (MASM) で作成した QB4.5 または BC7.1 用の Quick ライブラリは、そのままでは利用できません。Visual Basic for MS-DOS のリンカ (LINK.EXE) で再リンクする必要があります。また、Microsoft Macro Assembler のバージョンは 5.1 または 6.0 でなければなりません。新しく作成する Quick ライブラリは、VBDOSQLB.LIB とリンクします。Quick ライブラリの作成方法については、『プログラミングガイド』第 18 章を参照してください。

Quick ライブラリのコードで near 文字列のアドレスを使っているときは、far 文字列を使って書き直して、再アセンブルする必要があります。

4.3 オブジェクト モジュール ライブラリ (.LIB)

プログラミング環境で開発を行うときは、オブジェクト モジュール ライブラリを特に指定する必要はありませんが、Quick ライブラリと同時に作成したオブジェクト モジュール ライブラリを必ず用意してください。プログラミング環境起動時に Quick ライブラリをロードしておくと、[実行] メニューの [EXE ファイル作成] コマンドを選択するときに、適切なオブジェクト モジュール ライブラリがリンクされます。

コマンドラインからコンパイルおよびリンクを実行するときは、次のことに注意してください。

- Visual Basic for MS-DOS の開発環境上で Quick ライブラリを作成すると、Quick ライブラリに対応するオブジェクト モジュール ライブラリが自動的に作成されます。アプリケーションをリンクするときは、このライブラリを使います。
- Visual Basic for MS-DOS で提供されている Quick ライブラリを使うときは、同様に提供されている対応するオブジェクト モジュール ライブラリを使います。
- プログラミング環境外で開発を行うときは、far 文字列を使ってオブジェクト モジュール ライブラリを再コンパイルする必要があります (Visual Basic for MS-DOS の Basic コンパイラは near 文字列をサポートしていません)。
- QB4.5 では、浮動小数点の演算にデフォルトで代替数値演算ライブラリを使っていましたが、Visual Basic for MS-DOS ではエミュレータ ライブラリのみをサポートしています。

次の表は、コマンド ラインから実行可能 (.EXE) ファイルを作成するときに使う標準オブジェクト モジュール ライブラリについて、Visual Basic for MS-DOS と QB4.5 の対応を示しています。

QB4.5 のライブラリ	Visual Basic for MS-DOS のライブラリ
BCOM45E.LIB	VBDCL10.LIB
BCOM45A.LIB	Visual Basic for MS-DOS は代替数値演算ライブラリをサポートしていません。
BRUN45E.LIB	VBDRT10.LIB
BRUN45A.LIB	Visual Basic for MS-DOS は代替数値演算ライブラリをサポートしていません。

次の表は、コマンド ラインから実行可能 (.EXE) ファイルを作成するときに使う標準オブジェクト モジュール ライブラリについて、Visual Basic for MS-DOS と BC7.1 との対応を示しています。イタリック体で示してある文字は、数値演算パッケージまたは near/far 文字列の区別を示します。

BC7.1 のライブラリ	Visual Basic for MS-DOS のライブラリ
BCL71AFR.LIB	Visual Basic for MS-DOS は代替数値演算ライブラリをサポートしていません。
BCL71EFR.LIB	VBDCL10.LIB
BCL71*m*NR.LIB	Visual Basic for MS-DOS は near 文字列をサポートしていません。
BCL71*ms*P.LIB	Visual Basic for MS-DOS はプロテクト モードの実行可能ファイルをサポートしていません。
BRT71AFR.LIB	Visual Basic for MS-DOS は代替数値演算ライブラリをサポートしていません。
BRT71EFR.LIB	VBDRT10.LIB
BRT71*m*NR.LIB	Visual Basic for MS-DOS は near 文字列をサポートしていません。
BRT71*ms*P.LIB	Visual Basic for MS-DOS はプロテクト モードの実行可能ファイルをサポートしていません。

4.4 アプリケーションのロード

テキスト形式 (QB4.5 では "テキストファイル", BC7.1 では "Text") で保存されているソース ファイルであれば、Visual Basic for MS-DOS でそのままロードすることができます。バイナリ形式 (QB4.5 では "QuickBASIC 標準ファイル", BC7.1 では "Binary") で保存されているファイルは、テキスト形式で保存しなおしてください。

Visual Basic for MS-DOS と QB4.5 および BC7.1 では、一部の用語が異なります。QB4.5 および BC7.1 では、Basic のソース ファイル (.BAS) を "ファイル" と呼び、アプリケーションを構成するファイルの集合を "プログラム" と呼びます。Visual Basic for MS-DOS では、Basic のコード (.BAS) またはフォーム (.FRM) を含むソース ファイルを "モジュール" と呼び、アプリケーションを構成するモジュールの集合を "プロジェクト" と呼びます。次の表は、QB4.5 と Visual Basic for MS-DOS の [ﾌｧｲﾙ] メニューのコマンドの対応を示します。

QB4.5 の [ﾌｧｲﾙ] メニューのコマンド	Visual Basic for MS-DOS の [ﾌｧｲﾙ] メニューのコマンド
[新規]	[新規プロジェクト]
[読込]	[プロジェクト オープン]
[結合]	[テキスト読み出し]
[保存]	[ファイル保管]
[名前を変えて保存]	[新規保管]
[すべて保存]	[プロジェクト保管]
[サブファイル作成]	[新規フォーム] (.FRM ファイルの場合) [新規モジュール] (.BAS ファイルの場合)
[サブファイル読込]	[ファイル追加]
[サブファイル解放]	[ファイル解放]

QB4.5, BC7.1, Visual Basic for MS-DOS はいずれも、.MAK ファイルを使って複数のモジュールからなるアプリケーションを管理します。QB4.5 または BC7.1 では、.MAK ファイルの名前は実行開始ファイル (メイン モジュール) の名前によって決まり、プログラムをオープンするときに実行開始ファイルの名前を選択すると、必要なすべてのモジュールが自動的にロードされます。Visual Basic for MS-DOS では、プロジェクトをオープンするときに .MAK ファイルの名前を選択すると、すべての必要なモジュールがロードされます。

▼ QB4.5 または BC7.1 アプリケーションを Visual Basic for MS-DOS へロードするには

1. Visual Basic for MS-DOS のプログラミング環境で、[ﾌｧｲﾙ] メニューの [プロジェクト オープン] コマンドを選択します。
2. [プロジェクト オープン] ダイアログ ボックスで、ロードする QB4.5 または BC7.1 アプリケーションを選択します。アプリケーションが複数のモジュールからなるときは、.MAK ファイルを指定します。1 つのモジュールからなるアプリケーションでは、そのファイル名を入力して <了解> ボタンを選択します。

デフォルトでは、Visual Basic for MS-DOS のプログラミング環境で使うことができるメモリは QB4.5 よりも多くなりますが、BC7.1 よりはわずかに少なくなります。したがって、プログラミング環境のデフォルトの設定では、非常に大きな BC7.1 アプリケーションをロードできない場合があります。

▼ プログラミング環境で使うことができるメモリを増やすように設定するには

- VBDOS 起動時に /S オプションを指定します。

  例えば次のコマンドラインは、VBDOS.EXE が使うコンベンショナル メモリのサイズを約 300K に設定し、残りのメモリを ACCTREC プロジェクトのコードとデータで利用可能にします。

```
VBDOS /S:300 ACCTREC.MAK
```

プログラミング環境が使うコンベンショナル メモリのサイズは、/S で指定するバイト数と必ずしも一致しないことに注意してください。例えば /S:1 と設定すると、プログラミング環境が使うコンベンショナル メモリのサイズはもっとも小さくなりますが、プログラミング環境のサイズが 1K になるわけではありません。

/S スイッチの使い方について詳しくは、『プログラミング ガイド』の付録 B「メモリ管理」を参照してください。

4.5 予約語のチェック

Visual Basic for MS-DOS では、QB4.5 または BC7.1 より多くの単語が予約されています。QB4.5 または BC7.1 アプリケーション中で、定数，変数，ユーザー定義関数，ラベル，ユーザー定義型，プロシージャなどの名前として Visual Basic for MS-DOS で新しく追加された予約語を使っていると、構文エラーが起きます。このエラーを避けるには、新しく追加された予約語を使っていないかどうかをすべてのモジュールでチェックし、必要に応じて新しい名前に置き換えてください。

予約語の一覧は、『リファレンス』の付録 A またはオンライン ヘルプを参照してください。

4.6 ダイアログ ボックスの追加

Visual Basic for MS-DOS では、ウィンドウやダイアログ ボックスをフォームとして表示します。それぞれのフォームの内容は、拡張子 .FRM を持つ個別のファイルに記述されます。フォーム上には、コマンド ボタン，メニュー，入出力を行うときに使うテキスト ボックスなどの“コントロール”を配置します。またフォームには、ユーザーまたはシステムが行う動作 (マウスのクリックや画面の再描画など) によって発生する“イベント”に応答するコードが含まれます。

『プログラミング ガイド』では、フォーム，コントロール，イベントについてと、これらを組み合わせてイベントドリブン アプリケーションを作成する方法について説明しています。QB4.5 または BC7.1 のアプリケーションを Visual Basic for MS-DOS に移植するときに、フォームとコントロールを使ってアプリケーションを改良することには、次のような利点があります。

- フォーム デザイナを使うと、コントロールをフォームに配置していく設計作業だけで、コードを書かずに基本的なインターフェイスを構築できます。
- フォーム デザイナで配置するフォームやコントロールには、あらかじめ基本的な機能が組み込まれています。プログラマはイベントに対して処理を行うコードを書くだけでよく、イベントを認識するコードや、マウス カーソルの位置を確認するコードなどを記述する必要はありません。
- Visual Basic for MS-DOS では、コモン ダイアログ ツールキットを使うことができます。このツールキットを使うと、Visual Basic for MS-DOS 自身で使うダイアログ ボックスと同じような機能を持つダイアログ ボックス (ファイルのオープンや印刷など) を簡単に追加できます。

既存の QB4.5 または BC7.1 アプリケーションにフォームとコントロールを追加すると、ユーザー インターフェイスに関するコードを削減できます。例えば、ユーザー インターフェイスは次のような処理を行います。

1. ユーザーが選択可能な項目を決めます。
2. 選択項目と、それを囲むボックスを表示します。

3. ユーザーに応答を求めるメッセージ(「1, 2, 3 のいずれかを選択してください」など)を表示します。
4. ユーザーがキーを押したかどうかを監視します。
5. キーが押されたら、それが適切なキーであるかどうかをチェックし、間違ったキーであれば再度入力を求めます。
6. 適切なキーであれば、どのキーを押したかを判別し、対応する処理を呼び出します。
7. 対応する処理を行います。

一般に、ステップ 2 ～ 6 は Visual Basic for MS-DOS が自動的に行うため、プログラマが行わなければならないのはステップ 1 と 7 だけです。

再設計のレベル

QB4.5 または BC7.1 アプリケーションの Visual Basic for MS-DOS への移植は、次の 3 つの再設計レベルに分けて考えることができます。

- **変更しない**

  新しい機能をまったく使わない場合でも、QB4.5 アプリケーションを Visual Basic for MS-DOS で再コンパイルすると、一般にプログラムの性能が向上します (ただし浮動小数点演算を多用しているアプリケーションでは、数値演算パッケージの違いにより、必ずしも高速化しない場合があります)。再コンパイルの方法については、「コンパイルとリンク」を参照してください。

- **段階的な改良**

  大きなアプリケーションを扱うときや、既存のアプリケーションに新しく機能を追加するときは、1 度に少しずつ変更を行うこともできます。ただしこの場合は、アプリケーション全体の整合性に十分注意する必要があります。

- **完全な再設計**

  ユーザー インターフェイスを再設計すると、使い勝手が改善されるとともに、**PRINT** や **LOCATE** などのステートメントを使ってメニューやリストを画面に表示するときに比べ、書かなければならないコード量を削減できます。Visual Basic for MS-DOS の機能をフルに使ったアプリケーションにするためには、この方法を取る必要があります。

段階的な改良

既存の大きなアプリケーションを完全に再設計するには、当然かなりの時間が必要になります。ここでは、アプリケーションの一部を段階的に改良して行くことによって、アプリケーションに新しいインターフェイスを加えていく方法について説明します。

▼ アプリケーションへダイアログ ボックスを追加するには

1. 既存のアプリケーション中でユーザーインターフェイスの処理を行っているコードを探し、追加または置き換える部分を選択します。
2. **MSGBOX** と **INPUTBOX$** を使って、ユーザーとの単純な対話処理を行います。

   または

   コモン ダイアログ ツールキットを使って、ファイルのオープンまたは出力などの一般的なタスクを処理します。

   または

   フォーム デザイナを使って画面上にフォームを描き、イベント プロシージャにコードを追加してフォーム モジュールを作成します。フォームのロードと表示を行うコードを既存のアプリケーションに追加します。

これらそれぞれの方法について、順に説明して行きます。

インターフェイスに関するコード

QB4.5 または BC7.1 アプリケーションでは、多くの場合、アプリケーション中のコード全体にわたって、ユーザー インターフェイスの機能を実現するコードが書かれているはずです (ユーザー インターフェイスに関するコードは、一般にアプリケーションのコードの 50 ～ 60 パーセントを占めます)。QB4.5 または BC7.1 アプリケーションでは、通常次のようなステートメントや関数を使って、ユーザー インターフェイスを実現しています。プログラミング環境の [検索] メニューを使って、これらのステートメントや関数がアプリケーション中のどこで使われているかを確認してください。

- ユーザーからの可変長文字列または数値の入力を受け付ける **INPUT** ステートメント
- ユーザーからの固定長文字列や文字を受け付ける **INPUT$** および **INPUT¥** 関数
- 押されたキーを認識する **INKEY$** 関数
- 押されたキーに応じて処理を分岐する **KEY(*n*)** ステートメントと **ON KEY GOSUB** ステートメント
- ユーザーに応答を求めるときのプロンプトを表示したり、結果を表示したりするする **PRINT** ステートメント

Visual Basic for MS-DOS はこれらのステートメントや関数をすべてサポートしていますが、これらの代わりに、**MSGBOX** や **INPUTBOX$** 関数、またはコントロール (テキスト ボックス，ピクチャ ボックス，オプション ボタン，チェック ボックス，スクロール バー，リスト ボックスなど) を使うこともできます。QB4.5 または BC7.1 で **INPUT, INPUT$, INPUT¥, INKEY$, ON KEY** を使って行っていた処理のほとんどは、Visual Basic for MS-DOS の新しい機能で置き換えることができます。

次の表は、QB4.5, BC7.1 のステートメントおよび関数と Visual Basic for MS-DOS 機能の一般的な使い方を比較したものです。この表で、QB4.5, BC7.1 のステートメントおよび関数と Visual Basic for MS-DOS 機能の対応について理解してください。

QB4.5 または BC7.1 のステートメントおよび関数	動作/目的	Visual Basic for MS-DOS で同等の機能を実現する方法
INPUT ステートメント	キーボードから入力された可変長文字列または数値を返します。入力された文字は画面にも表示されます。	**INPUTBOX$** 関数またはテキスト ボックスを使います。
INPUT$ 関数, **INPUT¥** 関数	キーボードから入力された、指定された長さの文字列を返します。入力された文字は画面に表示されません。	キーボードからの入力を受け取るときに、入力された文字を画面に表示しない場合は、ピクチャー ボックスを使います。**MID$** または **KMID$** 関数を使って、必要な文字数を取り出します。
INKEY$ 関数	直前に押されたキーに対応する値を返します。	フォームまたはコントロールの KeyPress イベントに対するイベント プロシージャにコードを記述します。 特定の応答 (はい，いいえ，了解，取消など) を受け付けるには、**MSGBOX** 関数を使います。 オプション ボタンやチェック ボックスを使います。
KEY および**ON KEY GOSUB** ステートメント	あらかじめ指定してあったキーまたはキーの組み合わせが押されたときに、対応するサブルーチンに制御を移します。	QB4.5 または BC7.1 のサブルーチンのコードをフォームまたはコントロールの KeyPress イベントに対するイベント プロシージャに移動します。

INKEY$, INPUT$, INPUT¥, INPUT は、キーボードからの入力を受け付けるだけでなく、プロンプト (ユーザーに対するメッセージ) を表示できます。Visual Basic for MS-DOS では通常、プロンプトを表示するときにダイアログ ボックスを使います。ダイアログ ボックスは、次の 3 つ方法のうちのいずれかを使って表示します。

- **MSGBOX** 関数，**MSGBOX** ステートメント，**INPUTBOX$** 関数を使って、Visual Basic for MS-DOS があらかじめ用意しているダイアログ ボックスを表示します。
- コモン ダイアログ ツールキットに含まれている **SUB** プロシージャを呼び出します。
- 1 つのフォームとしてユーザー独自のダイアログ ボックスを設計し、そのフォームを表示します。

MSGBOX と INPUTBOX$ を使ったダイアログ ボックス

Visual Basic for MS-DOS は、アプリケーションにダイアログ ボックスをもっとも簡単に追加する方法として、**MSGBOX** 関数およびステートメントと，**INPUTBOX$** 関数を用意しています。これらを使うと、QB4.5 または BC7.1 アプリケーションで **INKEY$**, **INPUT$**, **INPUT¥** 関数や **INPUT** ステートメントを使って行っていた処理を比較的簡単に置き換えることができます。

Visual Basic for MS-DOS の関数およびステートメント	機能
MSGBOX 関数	ユーザーへのメッセージとコマンド ボタンを含むダイアログ ボックスを表示します。<はい>，<いいえ>，<了解>，<取消>，<中断>，<再試行>，<無視> のコマンド ボタンを配置できます。ユーザーがいずれかのボタンを選択すると、そのボタンを示す値を返します。
MSGBOX ステートメント	**MSGBOX** 関数と同様のダイアログ ボックスを表示しますが、値を返しません。
INPUTBOX$ 関数	ユーザーに対して入力を求めるメッセージとテキスト ボックス，コマンド ボタンを含むダイアログ ボックスを表示します。コマンド ボタンは <了解>，<取消> の 2 つです。ユーザーがテキスト ボックスにテキストを入力して <了解> ボタンを選ぶか [ENTER] キーを押すと、テキスト ボックスに入力された文字列を返します。ユーザーが <取消> ボタンを選ぶと、null 文字列 ("") を返します。

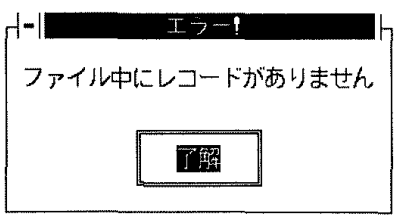
例えば、QB4.5 または BC7.1 アプリケーションで次のような処理を行っていたとします。

```
IF NumberOfRecords = 0 THEN
    PRINT: PRINT "ファイル中にレコードがありません。";
    PRINT "何かキーを押してください。";
    DO
    LOOP WHILE INKEY$ = ""
```

この処理は、Visual Basic for MS-DOS の **MSGBOX** 関数を使うと次のように書き換えることができます。

```
Res% = MSGBOX("ファイル中にレコードがありません。", 0, "エラー!")
```

この行を実行すると、画面中央に次のダイアログ ボックスが表示されます。

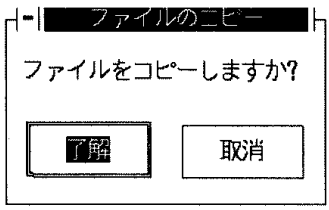


<了解> ボタンをクリックするか [ENTER] キーを押すと、ダイアログ ボックスは自動的にクローズします。

MSGBOX 関数の 2 番目の引数として 1 を指定すると、<了解> ボタンのほかに <取消> ボタンも表示され、ユーザーが操作を取り消すことができるようになります。<取消> ボタンがクリックされるか [ESC] キーが押されたときは、**MSGBOX** 関数は 2 を返します。

```
Res% = MSGBOX("ファイルをコピーしますか?", 1, "ファイルのコピー")
IF Res% = 2 THEN EXIT SUB   'ユーザーが操作を取り消したことを示します.
```

実行すると、次のダイアログ ボックスが表示されます。



次に示すのは、QB4.5 や BC7.1 プロシージャのコードです。この **SUB** プロシージャは、メニューを表示してドライブ名を選択するようユーザーに求め、ユーザーが選択するまで待ちます。ユーザーが選択したドライブ名は変数 TargetDrive$ に格納され、呼び出し側に戻されます。

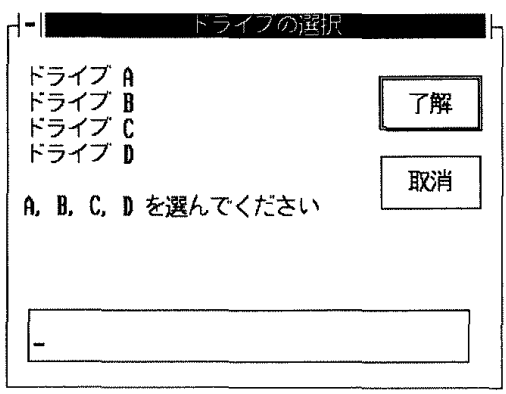
```
SUB GetDrive (TargetDrive$)
'メニューを表示するために, 独立した画面領域を作成します.
    VIEW PRINT 15 TO 22
    COLOR 7,0: CLS: COLOR 7,1
'メニューのバックグラウンドを表示します.
    FOR I% = 15 TO 22
        LOCATE I%,9:  PRINT SPACE$(64)
    NEXT I%
'メニュー テキストを表示します.
    J% = 15: LOCATE J%, 18: PRINT "ディスクの選択:"
    LOCATE J% + 2, 20:PRINT "ドライブ A"
    LOCATE J% + 3, 20:PRINT "ドライブ B"
    LOCATE J% + 4, 20:PRINT "ドライブ C"
    LOCATE J% + 5, 20:PRINT "ドライブ D"
    LOCATE J% + 7, 30:PRINT "A, B, C, D を選んでください"
'ユーザーがキーを押すまで待ちます.
ta2:
    DO
        A$ = INKEY$
    LOOP WHILE A$ = ""
'ユーザーが押したキーを評価します.
    A$ = UCASE$(A$)
    IF (A$ = "C" OR A$ = "D" OR A$ = "E" OR A$ = "F") THEN
        'ユーザーが押したキーが有効なドライブ名を表すときは, その文字を
        '呼び出し側に引き渡し, メニューをクリアします.
        TargetDrive$ = A$
        COLOR 7,0: CLS 2
```

```
        EXIT SUB
    ELSE
        GOTO ta2
    END IF
END SUB
```

このプロシージャは、ドライブ名を含むメッセージを表示し、ユーザーの応答を待ち、その応答を確認しますが、これは QB4.5 や BC7.1 でよく行われるプログラミングです。Visual Basic for MS-DOS でこのようなタスクを行うには、**INPUTBOX$** 関数を使います。

```
SUB GetDrive (TargetDrive$)
    DIM CRLF AS STRING
    CRLF = CHR$(13) + CHR$(10)
    A$ = ""
    Prompt$ = "ドライブ A" + CRLF
    Prompt$ = Prompt$ + "ドライブ B" + CRLF
    Prompt$ = Prompt$ + "ドライブ C" + CRLF
    Prompt$ = Prompt$ + "ドライブ D" + CRLF
    Prompt$ = Prompt$ + CRLF
    Prompt$ = Prompt$ + "A, B, C, D を選んでください"
    '<取消> ボタンを選択しても無効です.
    DO
        A$ = UCASE$(INPUTBOX$(Prompt$, "ドライブの選択",
          ⇒ TargetDrive$, 15, 9))
        IF (A$ = "A" OR A$ = "B" OR A$ = "C" OR A$ = "D") THEN
            TargetDrive$ = A$
            EXIT DO
        ELSE
            BEEP
        END IF
    LOOP
END SUB
```

DO ループ中の最初の行の **INPUTBOX$** 関数に注意してください。このプロシージャを実行すると、次のダイアログ ボックスが表示されます。このダイアログ ボックスを表示する位置は、**INPUTBOX$** 関数の 4 番目および 5 番目の引数で指定できます。



コモン ダイアログ ツールキットを使ったダイアログ ボックス

Visual Basic for MS-DOS は、ファイルのオープン，保管，印刷や文字列の検索と置換など、アプリケーションでよく使われるダイアログ ボックスを集めた、コモン ダイアログ ツールキットを用意しています。コモン ダイアログ ツールキットを使うと、プロシージャを呼び出すだけで、これらのダイアログ ボックスをアプリケーションに追加できます。詳しくは、『プログラミング ガイド』の第 5 章「ダイアログ ボックスの作成」や、第 10 章「ファイル システム コントロールの利用」を参照してください。

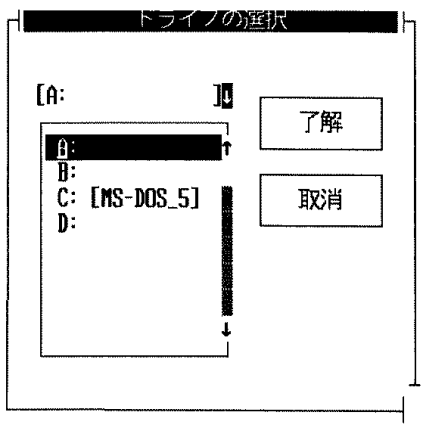
フォームを使ったダイアログ ボックス

アプリケーション独自のフォームを作成するには、フォーム デザイナを使ってフォームを設計します。この方法については、『プログラミング ガイド』の第 1 部を参照してください。ここでは、フォーム デザイナを使って、既存の QB4.5 または BC7.1 アプリケーションにフォームとコントロールを追加する方法を示します。

▼ QB4.5 または BC7.1 アプリケーションへ新しいフォームを追加するには

1. 画面上に表示するフォームをフォーム デザイナで作成し、フォーム モジュール (.FRM) として保管します。
2. フォームとフォーム上のコントロールを処理するコードをイベント プロシージャに追加します。
3. 新しいフォームのロードとアンロードを行うコードを既存のコード モジュール (.BAS) に追加します。
4. 必要に応じて、コード モジュールとフォーム モジュールの間で共有する変数を **COMMON SHARED** ステートメントで宣言します。宣言をインクルード ファイルにまとめて、両方のモジュールでインクルードすると便利です。

「MSGBOX と INPUTBOX$ を使ったダイアログ ボックス」では、**INPUTBOX$** を使ってドライブ名を取得する例を示しましたが、ここではドライブ リスト ボックスを使って、独自のダイアログ ボックスを作成する場合を考えてみます。



このフォームは、FormName プロパティに frmDrive という名前を設定して作成します。フォームを新規に作成すると、必ずフォーム モジュールが生成されることに注意してください。したがって、QB4.5 または BC7.1 アプリケーションにフォームを追加すると、必ず複数モジュールからなるアプリケーションになります。

まず、.BAS モジュールに次のコードを追加します。

```
'呼び出し側のモジュールのコード
'$FORM frmDrive
frmDrive.SHOW 1                    'フォームを表示します.
TargetDrive$ = LEFT$(frmDrive.Drive1.Drive, 1)
                                   'ドライブ名を取得します.
UNLOAD frmDrive                    'フォームをアンロードします.
```

さらに、フォーム モジュールに次のイベント プロシージャを追加します。

```
'フォーム モジュール (FRMDRIVE.FRM) のコード
SUB Command1_Click ()
    '<了解> を選択すると, フォームを非表示にします.
    Visible = 0
END SUB

SUB Command2_Click ()
    '<取消> を選択すると, ドライブをリセットしてフォームを非表示にします.
    Drive1.Drive = CURDIR$
    Visible = 0
END SUB
```

frmDrive.Drive1.Drive は、ドライブ リスト ボックス にそのとき設定されているドライブ名を示します。

このように適切なコントロールを利用すると、ほとんどの処理を Visual Basic for MS-DOS に自動的に行わせることができます。

非常に特殊な方法ですが、このようなテクニックは、フォームを表示しない状態でも利用できます。例えば、ファイル リスト ボックスだけを持つフォームを作成して次のようなイベント プロシージャを記述すると、フォームを表示しないまま、ファイル リスト ボックスの機能を利用できます。実行すると、カレント ディレクトリ内のすべてのファイルを、コマンドライン引数で指定したディレクトリにコピーします。

```
SUB Form_Load ()
    Visible = 0                'フォームは表示されません.
    'そのディレクトリにあるファイルの数だけ、コピーを繰り返します。
    FOR I% = 0 TO File1.ListCount - 1
        SHELL "COPY " + File1.List(I%) + " " + COMMAND$
    NEXT
END SUB
```

モジュール間での変数の共有

フォームを作成すると、1つのフォームについて1つのフォーム モジュールが生成されるため、既存のアプリケーションにフォームを追加した場合には、そのプロジェクトは必ず複数のモジュールで構成されることになります。この場合、モジュール間で変数を共有する必要が生じることがあります。

これを解決する最も単純な方法は、共有したい変数を宣言している **DIM** ステートメントを **COMMON SHARED** ステートメントで置き換えることです。モジュールレベル変数の宣言を **DIM** から **COMMON SHARED** に変更すると、その変数はモジュール内のすべてのプロシージャで参照できるようになります。しかし、この変更によってエラーが起きることもあります。変数を **COMMON SHARED** と宣言する前に、プロシージャ内でローカルに使われている変数に同じ名前のものがないかどうか、アプリケーションの各モジュールをチェックすることが重要です。プロシージャ内に同じ名前のローカル変数があるときは、プロシージャ レベルの変数の名前を変更するか、プロシージャ内で **DIM** によって宣言を行い、 **COMMON SHARED** 宣言を部分的に無効にします。

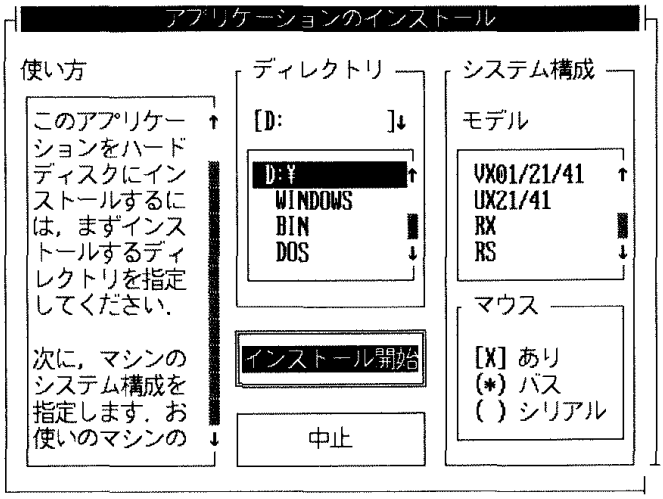
また、各モジュールで必ず同じ宣言をするためには、これらの宣言をインクルード ファイルにまとめ、各モジュールで **$INCLUDE** メタコマンドを使ってインクルードしてください。

4.7 インターフェイスの再設計

QB4.5 または BC7.1 アプリケーションの一部を変更しただけでは、Visual Basic for MS-DOS のすべての機能を利用することはできません。変更後のアプリケーションは、もとの QB4.5 または BC7.1 アプリケーションの構造を受け継ぎます。この状態では、ユーザーが入力を行うたびに新しいダイアログ ボックスが表示され、以前の選択内容は失われます。

Visual Basic for MS-DOS では、インターフェースを設計するときに Microsoft Windows のような方法を使い、コンテキスト (ユーザーが選択を行ったりデータを入力するときの画面上のトランザクション) 全体を保存できます。Visual Basic for MS-DOS のコントロールを有効に配置し、テキスト ボックスのスクロールやフォームのオーバーラップなどを利用すれば、ユーザーとの対話処理を直線的ではなく空間的に行うことができます。

イベントドリブン プログラミング モデルに基づいたユーザー インターフェイスは、これまでのような「情報を入力して、それが処理され、しばらく待ち、別の情報を入力して、それが処理され、またしばらく待つ」といったものではありません。次の図は単純な選択を組み合わせたユーザーインターフェイスを持つフォームで、インストールを開始する前にユーザーが選択する必要がある項目を示しています。このようにすれば、ユーザーは、デフォルトの設定を任意の順番で変更できると同時に、各項目の現在の設定値を確認しながら作業できます。


アプリケーションのインストール
使い方
このアプリケーションをハードディスクにインストールするには，まずインストールするディレクトリを指定してください．
次に，マシンのシステム構成を指定します．お使いのマシンの
ディレクトリ
[D:]
D:¥
WINDOWS
BIN
DOS
インストール開始
中止
システム構成
モデル
VX01/21/41
UX21/41
RX
RS
マウス
[X] あり
(*) バス
( ) シリアル

Microsoft® Visual Basic™
Programming System for MS-DOS®
機種依存情報
1993 年 2 月 1 日　初版第 1 刷発行

©1993 Microsoft Corp.

発行　マイクロソフト株式会社

Printed in Japan
落丁・乱丁はお取り替えいたします。

Microsoft®

046-100-901